НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity*

Виконав: студент IV курсу, групи *ІП-61 Кухарець Лілія Савеліївна* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Ліщук К.І.* _____
посада, науковий ступінь, вчене звання, прізвище, і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада, науковий ступінь, вчене звання, прізвище, і ім’я, по батькові (підпис)

Рецензент:

доц. каф. ТК., к.т.н. доц. Лісовиченко О.І.
посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
08.06.2020 16:24:16 EEST

Дата звіту:
08.06.2020 17:06:59 EEST

ID перевірки:
1003877376

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Kuharets_bachelor_ip61

ID файлу: 1003892356 Кількість сторінок: 77 Кількість слів: 8869 Кількість символів: 74551 Розмір файлу: 3.00 MB

6.63% Схожість

Найбільша схожість: 3.42% з джерело бібліотеки. ID файлу: 1000037970

4.26% Схожість з Інтернет джерелами	85	Page 79
5.93% Текстові збіги по Бібліотеці акаунту	307	Page 80

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	13

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Кухарець Лілії Савеліївній
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity*»
керівник проєкту *Ліщук Катерина Ігорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Вибір технологій програмування: визначення переваг обраних технологій
3) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
4) Розгортання та впровадження програмного забезпечення
5) Керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1) Схема структурна класів програмного забезпечення

2) Схема структурна варіантів використань

3) Схема структурна діяльності

4) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*05.05.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*10.05.2020*	
3.	*Постановка та формалізація задачі*	*12.05.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*15.05.2020*	
5.	*Алгоритмізація задачі*	*17.05.2020*	
6.	*Моделювання програмного забезпечення*	*18.05.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*20.05.2020*	
8.	*Розробка архітектури програмного забезпечення*	*21.05.2020*	
9.	*Розробка програмного забезпечення*	*26.05.2020*	
10.	*Налагодження програми*	*30.05.2020*	
11.	*Виконання графічних документів*	*03.06.2020*	
12.	*Оформлення пояснювальної записки*	*03.06.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*06.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Лілія КУХАРЕЦЬ
(підпис)

Керівник _____ Катерина ЛІЩУК
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-6115.045490.02.81	Пояснювальна записка	71	
3	А4	КПІ.ІП-6115.045490.03.91	Технічне завдання	11	
4	А4	КПІ.ІП-6115.045490.04.13	Опис програми	38	
5	А4	КПІ.ІП-6115.045490.05.51	Програма та методика тестування	7	
6	А4	КПІ.ІП-6115.045490.06.34	Керівництво користувача	8	
7	А3	КПІ.ІП-6115.045490.07.99.СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-6115.045490.07.99.СС	Схема структурна діяльності	1	
10	А3	КПІ.ІП-6115.045490.07.99.СС	Схема структурна класів програмного забезпечення	1	
10	А3	КПІ.ІП-6115.045490.07.99.КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6115. 045490.01.90		
Розробн.		Кухарець Л.С.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Ліщук К.І.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 141 сторінка, 25 рисунків, 17 таблиць, 5 додатків, 12 джерел.

Метою розробки мобільного тренажеру для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity є створення гри на смартфон, що дозволить підсилити інтерес школярів до вивчення фізики та стане одним із засобів дистанційного навчання.

У першому розділі проаналізовано предметну область та представлені на ринку аналоги, а також визначено вимоги до програмного забезпечення.

У другому розділі описано переваги мови програмування C# та ігрового рушія Unity.

У третьому розділі описано архітектуру програмного забезпечення та наведено подробиці реалізації проекту.

У четвертому розділі розроблено план тестування програмного забезпечення.

У п’ятому розділі описано процеси збірки та розгортання програмного забезпечення, а також подано інструкцію користувача.

Ключові слова: МОБІЛЬНИЙ ТРЕНАЖЕР, ФІЗИЧНІ СИЛИ, ІГРОВИЙ РУШІЙ UNITY, C#

ABSTRACT

The structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 141 pages, 25 figures, 17 tables, 5 appendices, 12 sources.

The purpose of developing a mobile simulator for studying physical force interactions using the Unity Game Engine is to create a game for a smartphone, which will increase students' interest in learning physics and will be one of distance learning tools.

In the first section the subject and analogues presented on the market are analyzed, also requirements to the software are defined.

The second section describes the benefits of the C# programming language and the Unity game engine.

The third section describes the software architecture and project implementation details.

In the fourth section a software testing plan is developed.

The fifth section describes the processes of software assembling and deploying, as well as user instructions.

Keywords: MOBILE SIMULATOR, PHYSICAL FORCES, UNITY GAME ENGINE, C#

Пояснювальна записка
до дипломного проєкту

на тему: Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

Слайдер – компонент графічного інтерфейсу користувача, що використовується для зміни значень.

Сетінг – середовище, в якому відбувається дія.

g – коефіцієнт прискорення вільного падіння.

UI – користувацький інтерфейс.

Скрол-бар – смуга прокрутки, елемент управління інтерфейсу.

Ігровий рушій – центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону [1].

View – представлення.

Model – модель.

Controller – контролер.

TextMeshPro – плагін для роботи зі шрифтами.

Zenject – інструмент ін’єкції залежностей, розроблений спеціально для Unity.

Linq –Language Integrated Query, запити, інтегровані в мову [2].

MVC – Model-View-Controller.

Scriptable Objects – скриптовий об’єкт.

Json – JavaScript Object Notation, запис об'єктів JavaScript.

Singleton – патерн одинак.

ВСТУП

Отримання шкільної освіти – невід’ємна частина життя кожної людини. Саме у школі дитина отримує базові знання, навички та вміння, які згодом використовуватиме у дорослому житті – для подальшого навчання, під час буденних ситуацій або навіть для проходження комп'ютерних чи мобільних ігор.

Комусь з учнів більше подобаються гуманітарні предмети, комусь – технічні. На жаль, деякі предмети багатьом залишаються незрозумілими. Так, учні звикли скептично ставитись до фізики, адже більшість вивченого матеріалу представлено лише теоретично.

Під час карантину шкільна освіта отримала ще один виклик – необхідність дистанційного навчання. За таких умов навіть проведення кабінетних експериментів стає неможливим.

На щастя, сучасні діти вміють та люблять працювати з технікою. Вирішення задач видається їм нудним, зате вони з задоволенням інтуїтивно прораховують траєкторії та швидкості об’єктів у іграх, навіть цього не помічаючи.

Використання гри, встановленої на смартфон, у якості навчального матеріалу має одразу декілька переваг. По-перше, це доступно. Сьогодні майже кожен школяр старших класів має сенсорний телефон. Школі не доведеться забезпечувати учнів технічними засобами, а учням – носити важкі ноутбуки до школи. По-друге, це цікаво. Візуалізація задач та система нагород за правильні рішення зроблять предмет не нудним і додадуть мотивацію. По-третє, ігри можна використовувати замість кабінетних експериментів в умовах дистанційного навчання.

На ринку представлені деякі рішення, що спонукають користувачів інтуїтивно пригадувати закони фізики. Але якщо проаналізувати їх з точки зору навчальної місії, можна знайти ряд недоліків. Такі застосунки не надають змоги розв’язати задачу формульно, без припущень. Окрім цього, деякі параметри в

таких іграх залишаються незмінними, наприклад, коефіцієнт прискорення вільного падіння.

Отже, створення тренажеру для вивчення взаємодії фізичних сил дозволить урізноманітнити вже існуючі ігрові механіки та зробити фізику менш нудною для учнів старших класів.

Розроблений застосунок зможе бути використаний у вигляді гри для дозвілля або як один з додаткових засобів шкільного навчання.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Змістовний опис і аналіз предметної області

Рух тіла, кинутого під кутом до горизонту – це складний криволінійний рух, який можна подати у вигляді суми двох незалежних рухів – рівномірного прямолінійного руху в горизонтальному напрямку та вільного падіння по вертикалі.

Якщо тіло кинути під кутом до горизонту, під час польоту на нього діятиме сила тяжіння та сила опору повітря. Зазвичай для вирішення шкільних задач силою опору нехтують, і залишається єдина сила – сила тяжіння. За другим законом Ньютона тіло рухається з прискоренням, рівним прискоренню вільного падіння, а проекції прискорення на вісі координат ax=0, ay=-g.

Проекції швидкостей тіла на осі матимуть наступний вигляд:

$\begin{cases} v_x = v_{x0} = v_0 cos\alpha \\ v_y = v_{y0} - gt = v_0 sin\alpha - gt \end{cases}$, де v_0 - початкова швидкість, а α - кут кидання [4].

Врахувавши час польоту, отримаємо координати тіла у кожен момент польоту:

$\begin{cases} x = vcos\alpha \cdot \Delta t \\ y = vsin\alpha \cdot \Delta t - \frac{gt^2}{2} \end{cases}$, якщо початковою координатою є (0; 0) [4].

Кут нахилу тіла у кожен момент польоту можна отримати, знайшовши тангенс цього кута шляхом ділення проекції швидкості на вісь y на проекцію швидкості на вісь x [4]:

$$tg = \frac{v_y}{v_x}$$

$$\beta = \text{atan}(tg)$$

Довжина польоту розраховується за наступними формулами [4]:

$$L = v_0 cos\alpha \cdot t = v_0 cos\alpha \cdot \frac{v_0^2 \cdot sin\alpha}{g} = \frac{v_0^2 \cdot sin2\alpha}{g}$$

Траєкторією руху тіла, кинутого під кутом до горизонту, є парабола.

Рисунок 1.1 - Рух тіла, кинутого під кутом до горизонту

Щоб задати стартову швидкість тіла, скористаємось поняттям імпульсу сили.

Імпульс сили – це фізична величина, що дорівнює добутку сили на час її дії, міра впливу сили на тіло протягом даного проміжку часу [5]:

$$I = F\Delta t$$

Імпульс сили та імпульс тіла є пов'язаними, адже імпульс сили рівний приросту імпульсу тіла під дією сили продовж певного часу.

$$I = \Delta p$$

Імпульс тіла – це міра механічного руху тіла, що рівна добутку маси тіла на його швидкість. Імпульс знаходимо за формулою [6]:

$$\Delta p = m\Delta v$$

Так, можемо представити імпульс сили наступним чином:

$$I = m\Delta v$$

Якщо початкова швидкість тіла дорівнює 0, то формула набуває вигляду:

$$I = mv_к$$

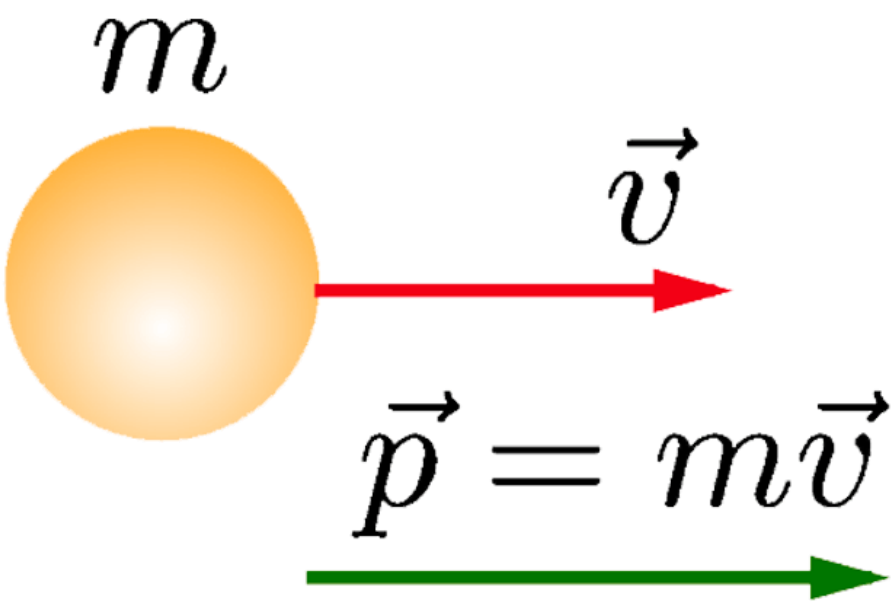


Рисунок 1.2 - Імпульс тіла

Похила площина – це плоска поверхня, встановлена під кутом до горизонталі.

За другим законом Ньютона:

$$ma = N + mg + F \text{ [7]}$$

Для тіла, що рухається вниз похилою площиною, прискорення розраховується за наступною формулою:

$a = g(sin\alpha + \mu \cdot cos\alpha)$, де α - кут нахилу площини, а μ - коефіцієнт тертя між площиною та тілом [7].

Якщо вважати, що рух тіла є рівноприскореним, зміна швидкості становитиме [7]:

$\Delta v = at$, де t - час руху

Для рівноприскореного руху координата тіла знаходиться за формулою [7]:

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

Якщо початкову координату вважатимемо за 0, а початкова швидкість є нульовою, формула координати тіла набуває вигляду:

$$x = \frac{1}{2}at^2$$

Знаючи координату кінцевої точки (тобто довжину похилої площини), можемо визначити час руху:

$$t = \sqrt{\frac{2l}{a}}$$

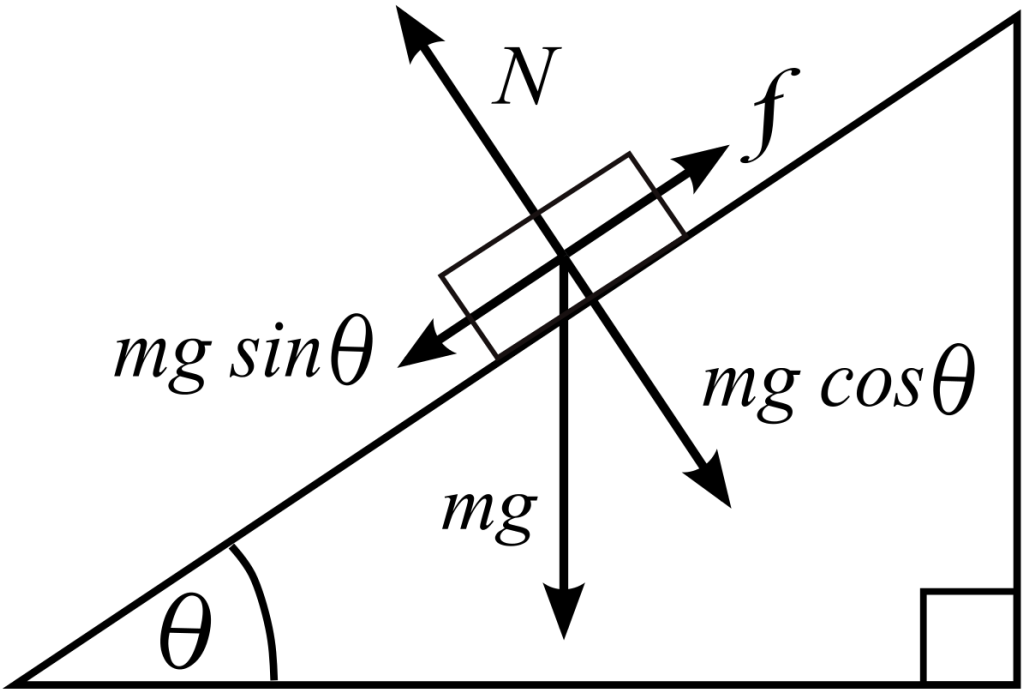


Рисунок 1.3 - Рух тіла похилою площиною

1.2 Аналіз відомих програмних продуктів

Хоча точних аналогів програмного продукту на продуктових платформах не представлено, існують ігри та застосунки на схожу тематику або з використанням подібних технологій.

Поширеною, але, на жаль, не дуже популярною серед користувачів є ідея створення фізичного калькулятора. Деякі розробники зосереджуються на конкретній області (механіка, електроніка, тощо), інші намагаються охопити весь курс шкільної фізики.

До застосунків-калькуляторів відноситься BetaPhysics. Основна ідея додатку – підбір формул на основі введеної користувачем умови.

Рисунок 1.4 - Фізичний кулькулятор BetaPhysics

Подібні застосунки мають аналогічні переваги та недоліки. Їх основним плюсом є демонстрація формул та прикладів розв’язання задач на велику кількість тем з курсу шкільної фізики. Серед недоліків таких застосунків – відсутність інтерактивної складової, що зацікавила б школяра. Хоча додатки оперують великою кількість формул, вони залишаються нудними.

Тренажер для вивчення взаємодії фізичних сил вирішує описану вище проблему. Користувач може візуалізувати розв’язання задачі та отримати заохочення до правильного розв’язання у вигляді доступу до нових ресурсів.

Крім того, існує можливість створення нових рівнів тренажеру, що описували б інші фізичні процеси. Подальшому розширенню функціоналу сприяє налагоджена система ін’єкції залежностей, наявність фабрики об’єктів та зручність збереження початкових ігрових даних засобами Unity.

Фізика руху тіла, кинутого під кутом до горизонту, зустрічається у іграх жанру головоломка. Користувачу необхідно поцілити снарядом у будівлю, супротивника або конкретну точку простору.

Популярними іграми жанру є Angry Birds, Worms та The Archers 2. У цих іграх користувач має прикласти до снаряду конкретну силу під певним кутом, аби влучити у ціль.

Рисунок 1.5 - Гра Angry Birds

Рисунок 1.6 - Гра Worms

Рисунок 1.7 - Гра The Archers 2

Такі ігри мають схожі переваги:

– зручне сенсорне направлення об’єкта;

– наявність яскравого сетінгу та рівнів;

– іноді зустрічається пунктирне відтворення траєкторії.

Сенсорне направлення має один суттєвий мінус – відсутня візуалізація показників, отже потрібної сили та кута досить важко досягнути. У мобільному тренажері ці проблеми вирішує слайдер – він дозволяє точно встановити потрібні параметри.

Рисунок 1.8 - Приклад слайдера, використаного у мобільному тренажері

Хоча тренажер не має яскравого сетінгу та сюжету, він пропонує поступове відкриття різних параметрів задачі шляхом збільшення кількості правильних розв'язків, аби урізноманітнити гру.

Пунктирне відтворення траєкторії є зайвим у тренажері, де траєкторія стане підказкою до розв’язання задачі.

Головоломки з застосуванням фізики тіла, кинутого під кутом до горизонту мають і недоліки у порівнянні з розробленим тренажером.

По-перше, це незмінне прискорення вільного падіння. У подібних іграх g залишається сталим. У мобільному тренажері g рандомно визначатиметься щоразу при завантаженні сцени;

По-друге, це інтуїтивна гра, відсутні обрахунки. Мобільний тренажер припускає як інтуїтивне рішення (додано допустиму похибку результату), так і рішення шляхом розв’язання задачі.

1.3 Аналіз вимог до програмного забезпечення

1.3.1 Розроблення функціональних вимог

Для створення use-case діаграми визначимо акторів, що взаємодіятимуть з програмним забезпеченням. Так як застосунок є грою для смартфону, єдиним учасником системи є власник гаджету, тобто користувач.

Рисунок 1.9 - Схема структурна варіантів використання

В таблиці 1.1 подано основні функціональні вимоги до програмного забезпечення.

Таблиця 1.1 – Основні функціональні вимоги до ПЗ

Код вимоги	Опис вимоги
R1	Вибір міні-гри з головного меню
R2	Функціонування міні-гри про рух тіла під кутом до горизонту
R2.1	Випадковий вибір початкових параметрів середовища (прискорення вільного падіння, маса тіла)
R2.2	Випадковий вибір цільової точки, якої має досягнути об’єкт
R2.3	Зміна сили, прикладеної до об’єкта та кута її прикладання
R2.4	Рух об’єкту за формулами руху тіла, кинутого під кутом до горизонту
R3	Функціонування міні-гри про рух тіла похилою площиною
R3.1	Випадковий вибір початкових параметрів середовища (коефіцієнт тертя)
R3.2	Випадковий вибір швидкості, якої об’єкт має досягнути в кінці площини.
R3.3	Зміна кута нахилу похилої площини та маси об’єкта
R3.4	Рух об’єкта за формулами руху тіла похилою площиною
R4	Підрахунок програшів та перемог
R4.1	Збереження програшів та перемог у пам’яті пристрою
R5	Доступ до параметрів середовища залежно від виграшів

Варіанти використання ПЗ подано у таблиці 1.2.

Таблиця 1.2 – Варіанти використання ПЗ

Код	Варіант використання	Опис
Вибір гри	UC1	Вибір міні-гри (тренажеру) з головного меню.
Гра в тренажер про рух тіла, кинутого під кутом до горизонту	UC2	Завантаження рівня з грою, зміна сили, що прикладається до ракети та кута нахилу, запуск ракети.
Гра в тренажер про рух тіла похилою площиною	UC3	Завантаження рівня з грою, зміна маси тіла на площині та кута нахилу площини, відпускання ракети.
Перегляд програшів та перемог	UC4	Перегляд прогресу у міні-іграх у вікні профілю.
Перегляд доступних параметрів середовища	UC5	Перегляд доступних планет, ракет та матеріалів площини у відповідних вікнах.

	UC1	UC2	UC3	UC4	UC5
R1	■	■	■		
R2		■			
R2.1		■			
R2.2		■			
R2.3		■			
R2.4		■			
R3			■		
R3.1			■		
R3.2			■		
R3.3			■		
R3.4			■		
R4				■	
R4.1				■	
R5					■

Рисунок 1.10 - Матриця залежності між функціональними вимогами та варіантами використання

1.3.2 Розроблення вимог до UI

Оскільки тренажер складається з декількох ігор, в процесі розробки потрібно створити:

- сцену головного меню;
- сцену гри про рух тіла, кинутого під кутом до горизонту;
- сцену гри про рух тіла похилою площиною.

Перелік вимог до UI:

– на екрані головно меню розмістити кнопки переходу в ігри, перегляду профілю гравця, перегляду доступних ігрових об’єктив-параметрів;

– перегляд доступних ігрових об’єктив-параметрів реалізувати у вигляді горизонтальних скрол-барів;

– на екрані профілю розмістити дані про кількість програшів та виграшів у іграх;

– на екрані гри про рух тіла під кутом до горизонту розмістити слайдери для зміни сили, прикладеної до тіла та кута польоту, космічний човен, вказівник бажаного місця приземлення, відомі параметри задачі у вигляді текстових полів та кнопку початку руху;

– на екрані гри про рух похилою площиною розмістити слайдери для зміни кута нахилу та маси тіла, площину з тілом, відомі параметри задачі та бажаний результат у вигляді текстових полів, кнопку початку руху.

1.4 Постановка завдання

1.4.1 Призначення розробки

Розробка призначена для вирішення задач на взаємодію фізичних сил у формі гри. Вона надає можливість користувачу коротати час за проходженням рівнів зі змінними параметрами або поглиблювати знання зі шкільної програми фізики. Розробка включає 2 гри на взаємодію різних фізичних сил та систему відкриття нових параметрів залежно від перемог у іграх.

1.4.2 Цілі та задачі розробки

Метою даної розробки є посилення інтересу школярів до вивчення фізики, їх мотивування правильно вирішувати задачі, а також розширення асортименту існуючих засобів дистанційного навчання. Крім того, розроблене ПЗ призначене для розширення наявних ігрових механік, зокрема додавання змінного коефіцієнту вільного падіння для імітації руху тіла, кинутого під кутом до горизонту.

Для досягнення поставленої мети необхідно вирішити комплекс взаємопов’язаних задач:

– при вході у гру завантаження головного екрану з відображенням іконок міні-ігор;

– перегляд доступних параметрів ігрового середовища у вигляді скролу;

– перегляд програшів та перемог для кожної міні-гри у вікні профілю;

– розробка міні-гри про рух тіла, кинутого під кутом до горизонту;

– розробка міні-гри про рух тіла похилою площиною;

– генерація рандомних параметрів задачі при завантаженні сцени гри;

– відтворення руху тіла за фізичними законами після натискання клавіші старту;

– збереження програшу чи виграшу після припинення руху тіла.

1.5 Висновки по розділу

У даному розділі описано предметну область розробки, зокрема подано основні фізичні формули та поняття. Також досліджено наявні на ринку аналоги. Зважаючи на виявлені недоліки існуючих програм, можна дійти висновку, що створення мобільного тренажеру вирішує основну проблему фізичних калькуляторів – відсутність візуалізації.

2 ВИБІР ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ

2.1 Вибір ігрового рушія

Unity Engine – це середовище розробки ігор під різні платформи. Unity був випущений у 2005 році американською компанією Unity Technologies та продовжує вдосконалюватись навіть сьогодні [8].

Unity дозволяє створювати додатки для комп'ютерів (під Windows, MacOS та Linux), смартфонів (iOS, Android та Windows Phone). Можливе написання веб-застосунків та ігор для консолей.

Ігровий рушій є дуже популярним – у App Store та Google Play можна знайти безліч додатків, створених з його допомогою. Крім того, Unity використовується для навчання основам створення ігор, оскільки є безкоштовним для некомерційних проектів.

Рушій має досить низький поріг входження, чудову документацію та багато відео-гайдів на пов’язану тематику, що дозволяють швидко освоїти основи та продовжити поглиблювати знання пізніше. Прості ігри можна створювати, не написавши жодного рядка коду.

Для більш складних додатків Unity пропонує підтримку C#, JavaScript та Boo. Фізичні розрахунки виконуються за допомогою PhysX (фізичний движок, розроблений NVIDIA).

Інтерфейс Unity базується на принципі Drag & Drop, тобто користувачу достатньо перетягувати елементи у конкретні місця в ієрархії інших компонентів, аби створювати взаємозв'язки між ними.

Unity підтримує майже все: фізику твердих тіл, систему Level of Detail, колізії між об’єктами, анімації, інтерактивний інтерфейс з кнопками, слайдерами та скролами, звуковий супровід, світло, тощо.

Класичний проект, розроблений на Unity, поділяється на сцени. Кожна сцена – це окремий світ зі своїми зв'язками та залежностями. На сцені можуть бути розміщені об’єкти у просторі, а також канви UI об’єктів. Сцена

відтворюється за допомогою камери, яких може бути і декілька, але не менше однієї. Кожна камера показує власний набір об’єктів.

До об’єкту можна приєднати різноманітні компоненти, серед яких розміщення у просторі, фізичні рамки, зображення, тощо. Одним із компонентів є і код, написаний користувачем.

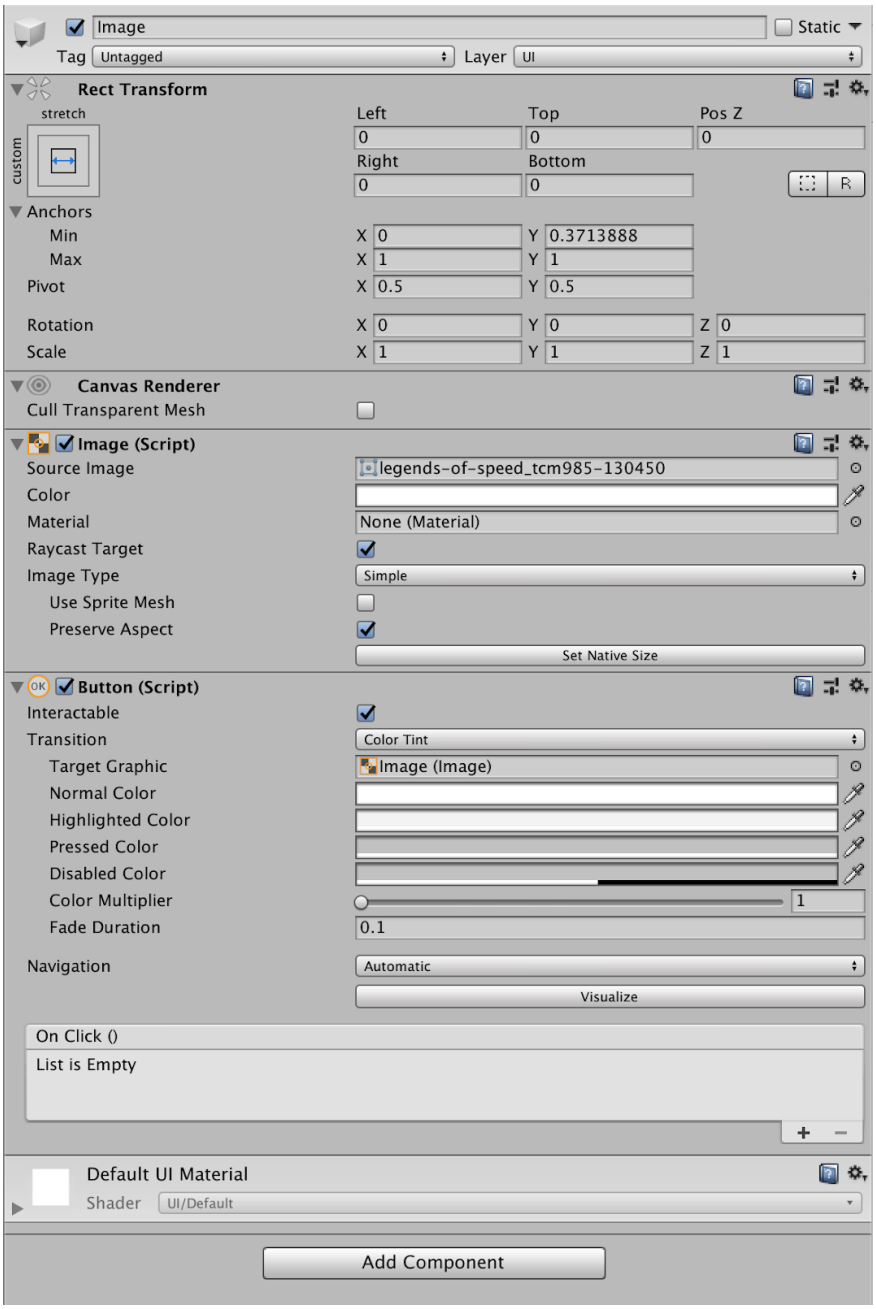


Рисунок 2.1 - Приєднання компонентів до об’єкту у Unity

Unity пропонує зручну ієрархію об’єктів – так, дочірні об’єкти будуть змінюватись разом з батьківськими.

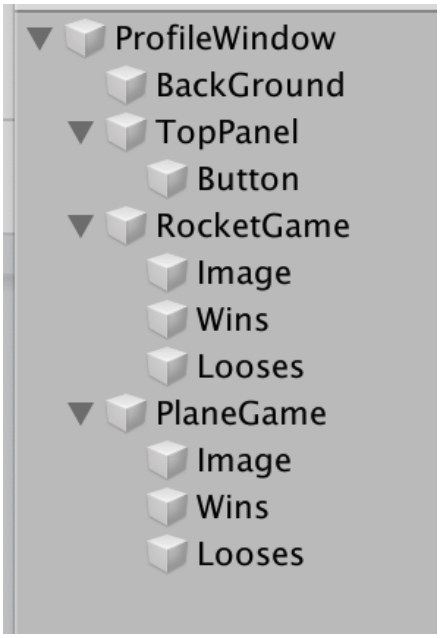


Рисунок 2.2 - Ієрархія об’єктів у Unity

Ігровий рушій Unity обрано для розробки мобільного тренажеру, зважаючи на його наступні особливості:

– колаборація з C #. Написання власного коду дозволяє значно розширити базовий функціонал рушія. В свою чергу, Unity також розширює наявний функціонал мови;

– кросплатформеність. Єдиний код та інтерфейс може бути зібраний у вигляді гри під будь-які девайс або платформу. До того ж, Unity дозволяє використовувати Scripting Define Symbols – текстові змінні, введення яких у настройках дозволяє замінити одну відмічену частину коду на іншу. Така функція уможливлює використання різних плагінів для різних платформ;

– має спеціальний сервіс Asset Store, де розміщені платні та безкоштовні набори анімацій, зображень, UI елементів, шрифтів тощо для підвантаження у проект. Для інді-розробників, що працюють без художників чи аніматорів, така особливість є дуже важливою;

– наявність базового класу MonoBehaviour. Дочірні класи MonoBehaviour можуть бути прикріплені до об’єктів у редакторі та взаємодіяти зі звичайними C

класами. Це робить можливим зміну об’єктів під час гри, коригування параметрів класу з редактора без переписування коду, тощо. У проекті мобільного тренажеру ця особливість дозволяє реалізувати MVC архітектуру з повним розділенням View та Model-Controller частин;

– можливість збереження даних у вигляді Scriptable Objects. Скриптові об’єкти – це екземпляри класу, створені не з коду, а в редакторі. За допомогою Scriptable Objects можна задати та налаштувати ієрархію даних, що трохи схожа на базу даних. Усі екземпляри скриптових об’єктів можна підвантажити у код, якщо вони розташовані у спеціальній папці Resources;

– великий вибір UI елементів. Для реалізації інтерфейсних задач тренажера необхідні кнопки, слайдери, скроли та текстові поля. Усі ці елементи Unity підтримує у декількох екземплярах;

– можливість завантажити додаткові плагіни. Так, у проекті використано плагін для створення красивих шрифтів – TextMeshPro;

– наявність зручного інструменту ін’єкції залежностей Zenject, розробленого специально для Unity.

2.2 Вибір мови програмування

C# – сучасна об’єктно-орієнтована мова програмування. C# належить до сімейства мов C та має схожий синтаксис з C, C++ чи Java [10].

Хоча C# є об’єктно-орієнтованою мовою, вона також підтримує компонентно-орієнтоване програмування.

C# пропонує декілька функцій, що забезпечують надійність та стійкість додатків. Наприклад, так зване прибирання сміття автоматично звільняє пам’ять, яку займають об’єкти, що вже не використовуються та є недосяжними з інших областей коду. Обробка виключень пропонує структурований та легко розширювальний підхід до виявлення помилок та їх обробки. Оскільки мова забезпечує безпеку типів, читання неініціалізованих змінних стає неможливим,

користувачу забороняється вихід за межі масиву, приведення типів має бути перевіреним.

C# пропонує єдину систему типів. Усі типи, навіть примітиви, є нащадками object, отже, мають загальний набір методів та подібний спосіб збереження і обробки.

C# підтримує створення власних значимих та посилальних типів даних. Таким чином, користувач сам вирішує, як виділити пам’ять для об’єкту – динамічно у купі чи у стеці.

Оскільки мова продовжує розвиватись і у нас час, багато уваги було приділено контролю версій. Це вплинуло на роздільні модифікатори virtual та override, правила перевантаження методів та дозвіл явно оголошувати елементи інтерфейсів.

Нові версії мови використовують інші парадигми програмування. Так, C# підтримує лямбда-вирази (елемент функціонального програмування).

Мову C# обрано для розробки мобільного тренажеру, зважаючи на наведені нижче особливості.

– Об’єктно-орієнтований підхід. Особливість дозволяє працювати з абстракціями. Під час розробки ігор з MVC архітектурою об’єктно-орієнтований підхід спрощує зв’язок між представленням та контролером. Контролер звертається до абстрактного представлення, яке в свою чергу делегує виконання окремих функцій іншим класам, відповідним за UI. В свою чергу, UI може функціонувати без готової моделі, якщо замість абстракції використати сет константних даних;

– Наявність синтаксичного цукру. C# пропонує багато вже готових рішень, що економлять час програміста і роблять код максимально зрозумілим.

– Велика кількість бібліотек, що можуть бути підвантажені до проекту. Так, під час розробки диплому часто виникала потреба у використанні засобів бібліотеки Linq, зокрема методів пошуку за атрибутом та сортування. Математичні обрахунки виконувались за допомогою функцій бібліотеки Math.

– Зручна реалізація методів розширення. У C # вони реалізовані як статичні методи в статичних класах, з першим аргументом типу розширюваного класу з ключовим словом this. Такий підхід дозволяє винести всі розширення у окремі файли та спрощує звернення до них.

```
public static T GetRandom<T>(this List<T> elements)
{
    return elements[elements.Count > 1 ? UnityEngine.Random.Range(0, elements.Count) : 0];
}
```

Рисунок 2.3 - Метод розширення, написаний мовою C #

– Колаборація з Unity. Хоча Unity підтримує і JavaScript, сумісність з C# вища. C# має більше можливостей для розробки гри, більше пов’язаних ресурсів для навчання, він забезпечує кращу продуктивність у колаборації з ігровим движком . Плагіни, необхідні для розробки тренажера, також написані на мові C#.

2.3 Висновки до розділу

У розділі подано інформацію про обрані технології розробки – ігровий рушій Unity та мову програмування C#. Дані засоби використані для розробки, зважаючи на підтримку рушієм мови, зручні засоби для створення користувацького інтерфейсу Unity та об’єктно-орієнтований підхід C#.

3 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Моделювання та аналіз програмного забезпечення

Основним процесом даного мобільного застосунку є вирішення фізичних задач у формі гри. Дії користувача та відповідні реакції програми описано за допомогою наступного алгоритму:

Рисунок 3.1 - Схема структурна діяльності для гри про політ тіла під кутом до горизонту

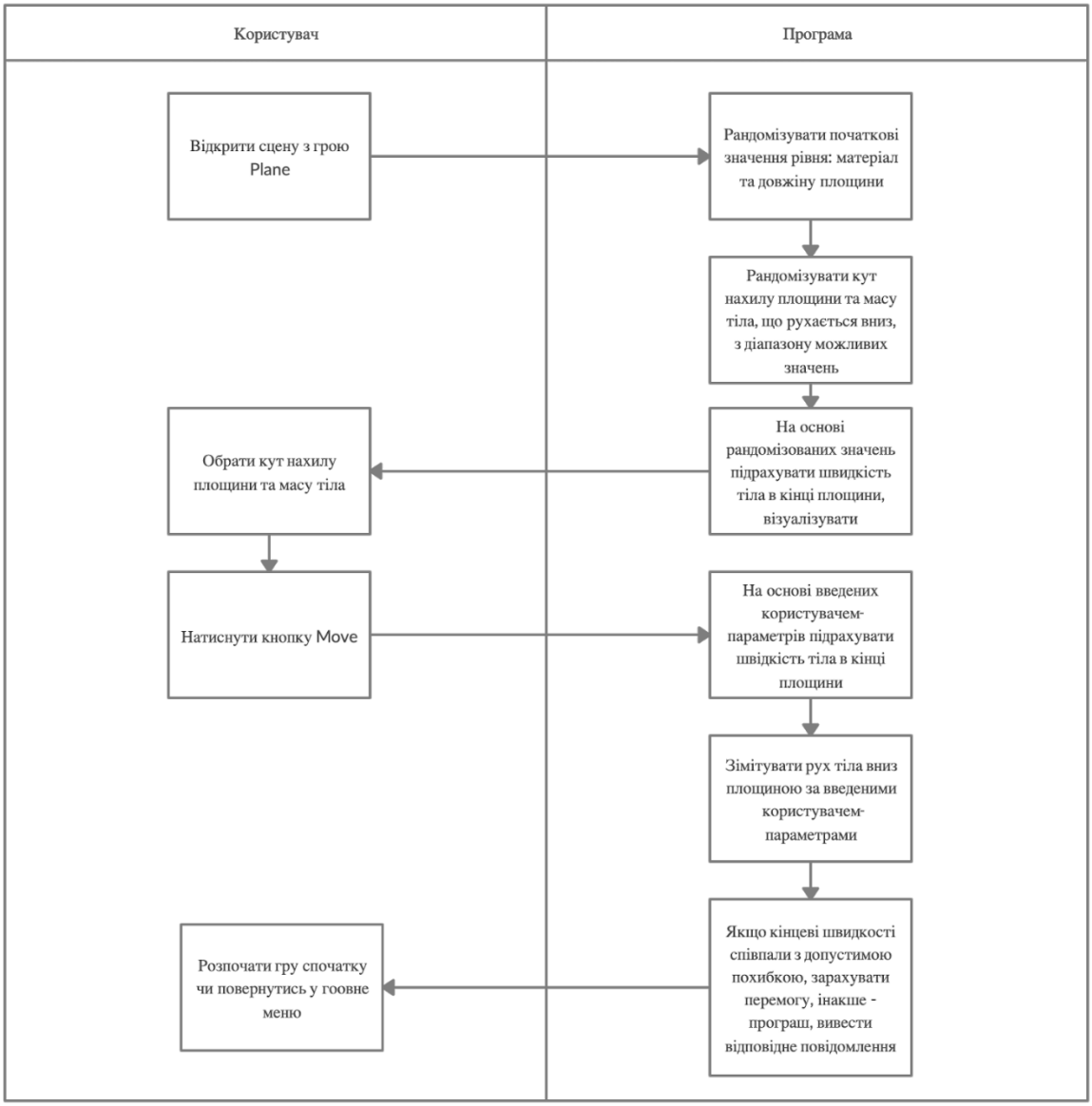


Рисунок 3.2 - Схема структурна діяльності для гри про рух тіла похилою площиною

а) користувач відкриває гру, програма у відповідь створює екран головного меню;

б) користувач обирає гру шляхом натискання відповідної кнопки, програма завантажує потрібну сцену та створює екран гри;

в) відбувається програмна рандомізація початкових параметрів рівня та їх відтворення в UI;

г) рандомізуються параметри задачі рівня та за допомогою фізичних формул обраховується відповідь, що відтворюється в UI як ціль, якої має прагнути користувач;

д) користувач взаємодіє з UI, змінюючи параметри задачі таким чином, аби його розв’язок співпав з запропонованим програмою та натискає кнопку руху;

е) програма відтворює рух тіла за допомогою фізичних формул та обраховує розв’язок на основі параметрів, введених користувачем;

є) програма порівнює власний розв’язок та розв’язок користувача і якщо вони співпадають з допустимою похибкою, виводить повідомлення про виграш, інакше – про програш;

ж) користувач може зіграти ще раз або повернутись у головне меню.

Менш суттєвим є процес отримання доступу до нових об’єктів (ракет, планет, матеріалів площини) зі збільшенням кількості виграшів. Гравець може переглянути усі наявні елементи, відкривши відповідний екран з головного меню. Якщо елемент доступний (потрібна для його відкриття кількість перемог більша за наявну), вся інформація про нього буде розшифрована, інакше – зашифрована.

3.2 Архітектура програмного забезпечення

Програмне забезпечення реалізоване за допомогою MVC архітектури [10], підлаштованої під особливості розробки мобільних ігор.

Дані, яки гра отримує на вхід зберігаються у вигляді скриптованих об’єктів (Scriptable Objects).

Зі сторонніх плагінів використовуються Json.Net Unity та Zenject Unity.

3.2.1 MVC архітектура

Кожному екрану або вікну поставлено у відповідність окрему View (відповідний скрипт прикріплено до префабу).

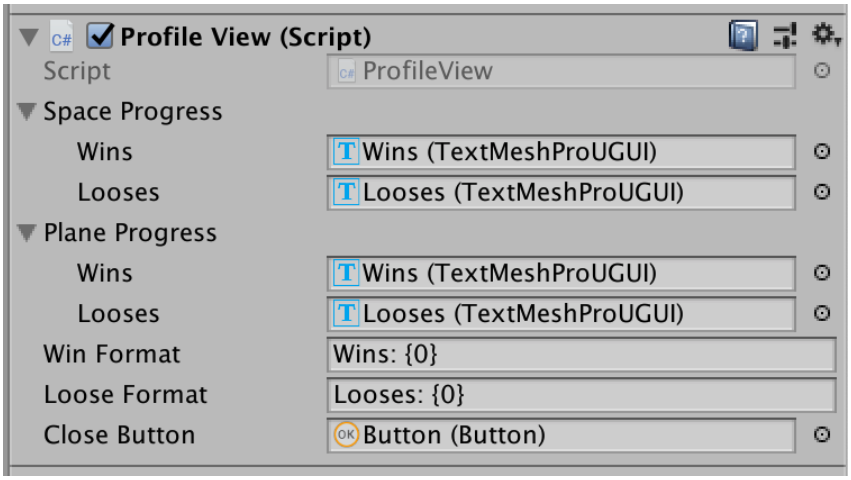


Рисунок 3.3 - Вигляд скрипта представлення з редактора

Unity дозволяє зручно зв’язувати елементи UI (складові префаба) та елементи класу.

```
public class RetryWindow:WindowView
{
    [SerializeField]
    private TMP_Text _text;

    [SerializeField]
    private Button _retryButton;

    [SerializeField]
    private Button _closeButton;
```

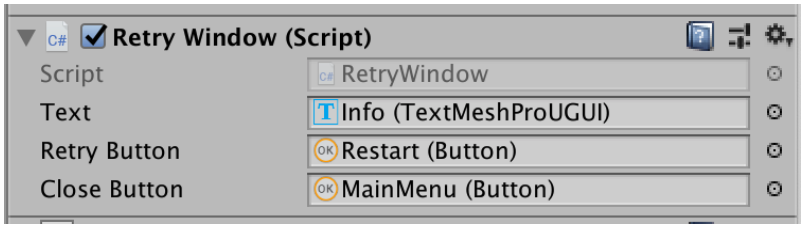


Рисунок 3.4 - Приклад зв’язування елементів UI та елементів класу.

View відображає дані та реагує на взаємодію користувача з UI. Усі класи даного типу є нащадками класу WindowView, що наслідується від MonoBehaviour.

View не знає про модель чи контролер. Але вона публікує події, на які підписаний контролер при конкретних діях користувача (натисканні кнопки, зміні значення слайдера тощо).

```
[SerializeField]
private Button _closeButton;

public Action onClose;

private void Start()
{
    _closeButton.onClick.AddListener(() => onClose?.Invoke());
}
```

Рисунок 3.5 - Виклик події при натисканні на кнопку

Контролер містить посилання на відповідну йому View та класи моделі, він може змінювати View та дані моделі, а також ініціювати відкриття нових вікон.

```
private void OpenWindow(WindowType type)
{
    IController controller = _windowPool.Get(type);
    controller.Open();
}
```

Рисунок 3.6 - Відкриття вікна з класу HomeController

В конструкторі контролер отримує посилання на всі інструменти, з якими він має працювати та вибирає з пулу потрібну View. Тут же відбувається початкова ініціалізація даних та підписка на події. Контролер може звернутись до відповідного йому представлення чи моделей напряму. Хоча контролери не є нащадками MonoBehaviour, в них можна використовувати корутини за допомогою допоміжного класу AsyncProcessor. Контролер реалізує інтерфейс IController.

Моделі відповідають за завантаження ігрових даних та їх зберігання протягом ігрової сесії, зміну прогресу гравця, фізичні обрахунки.

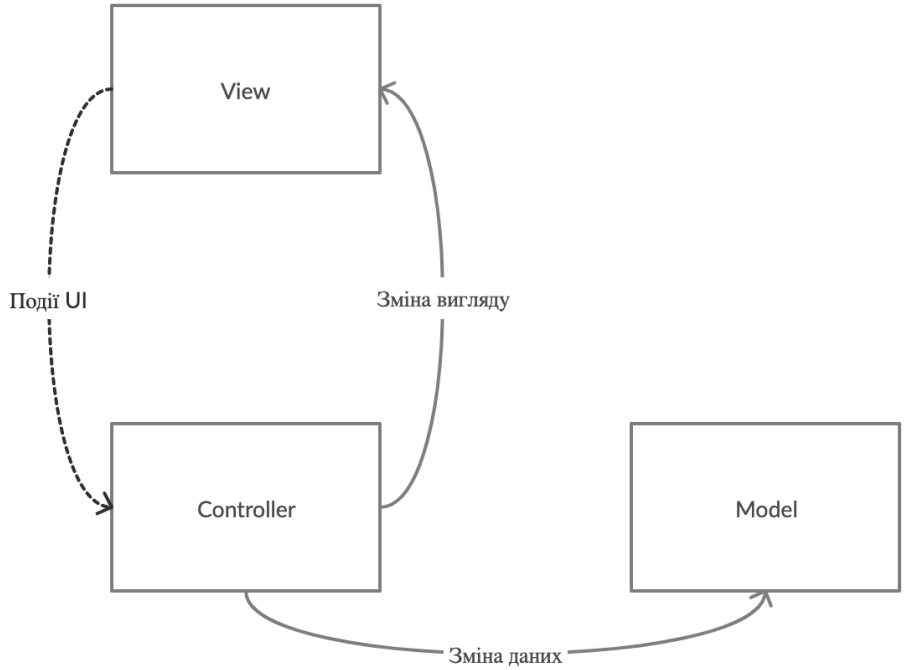


Рисунок 3.7 - Схема архітектури, використаної під час розробки

3.2.2 Збереження даних

Вхідні дані проекту, що підтягуються класами моделі, зберігаються у вигляді скриптованих об’єктів (Scriptable Objects). Така форма зберігання дозволяє створювати і редагувати об’єкти з редактора без необхідності змінювати код.

```
[CreateAssetMenu(menuName = "ScriptableObject/Space/Planet")]
public class PlanetInfo : ScriptableObject
{
    [SerializeField]
    private int _wins;

    [SerializeField]
    private string _planetName;

    [SerializeField]
    private SpriteView _surface;

    [SerializeField]
    private SpriteView _sky;

    [SerializeField]
    private SpriteView _icon;

    [SerializeField]
    private float _g;

    public string planetName { get { return _planetName; } }

    public SpriteView surface { get { return _surface; } }

    public SpriteView sky { get { return _sky; } }

    public SpriteView icon { get { return _icon; } }

    public float g { get { return _g; } }

    public int wins { get { return _wins; } }
}
```

Рисунок 3.8 - Клас-нащадок ScriptableObject

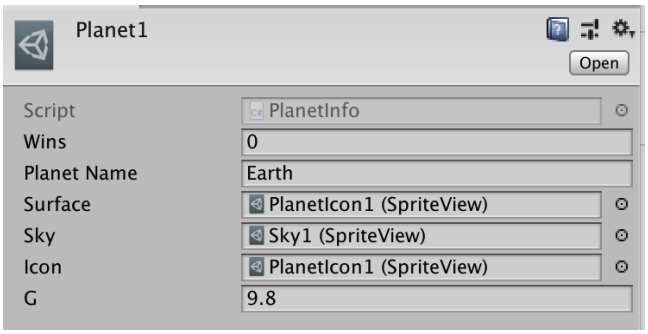


Рисунок 3.9 - Налаштування скриптованого об’єкта з редактора

Детально опишемо структуру скриптованих об’єктів, використаних в проекті.

Таблиця 3.1 - Короткий опис класів скриптованих об’єктів

Назва класу (назва об’єкту)	Опис
GameTexts (GameTexts)	Список текстів, що використовуються у попапах
PrefabsInfo (Prefabs)	Список префабів, копії яких створюються під час гри
ScalesInfo (Scales)	Список параметрів масштабування в залежності від розширення екрану пристрою
SpriteView (SpriteView)	Збереження зображення
PlaneInfo (PlaneInfo)	Параметри задачі про рух похилою площиною
PlaneMaterial (PlaneMaterial)	Параметри матеріалу площини
PlanetInfo (Planet)	Параметри планети
SpaceshipInfo (Spaceship)	Параметри космічного човна
SpaceInfo (SpaceInfo)	Параметри задачі про рух тіла, кинутого під кутом до горизонту

Таблиця 3.2 - Опис об’єкту GameTexts

Поле	Тип	Призначення
gameTexts	List<GameText>	Список текстів, що використовуються у попапах
Поля класу GameText		
type	TextType	Тип тексту
phrase	string	Власне текст

Таблиця 3.3 - Опис об’єкту Prefabs

Поле	Тип	Призначення
prefabs	List<BasePrefab>	Список посилань на префаби, де BasePrefab – батьківський клас для класів, що пов’язані з префабами, копії яких створюються під час гри

Таблиця 3.4 - Опис об’єкту Scales

Поле	Тип	Призначення
scales	List<ScaleInfo>	Список параметрів масштабування
Поля класу ScaleInfo		
sizeLimits	Vector2	Ліміти відношення висоти екрану до ширини
rocketScale	RocketScale	Параметри масштабування для гри про рух тіла, кинутого під кутом до горизонту
planeScale	PlaneScale	Параметри масштабування для гри про рух похилою площиною

Таблиця 3.5 - Опис об’єкту SpriteView

Поле	Тип	Призначення
sprite	Sprite	Посилання на зображення, що використовується в проекті

Таблиця 3.6 - Опис об’єкту PlaneInfo

Поле	Тип	Призначення
g	float	Коефіцієнт прискорення вільного падіння
minLength	int	Мінімальна довжина похилої площини
maxLength	int	Максимальна довжина похилої площини
minMass	int	Мінімальна маса об’єкту, що рухається похилою площиною
maxMass	int	Максимальна маса об’єкту, що рухається похилою площиною
minAngle	int	Мінімальний кут нахилу похилої площини
maxAngle	int	Максимальний кут нахилу похилої площини
finalSpeedLimits	Vector2	Допустима похибка відхилення кінцевої швидкості при визначенні програшу чи виграшу

Таблиця 3.7 - Опис об’єкту PlaneMaterial

Поле	Тип	Призначення
materialName	string	Назва матеріалу площини
view	SpriteView	Зображення матеріалу
icon	SpriteView	Іконка матеріалу
mu	float	Коефіцієнт тертя для матеріалу та дерева
wins	int	Кількість перемог у грі, необхідна, щоб відкрити матеріал

Таблиця 3.8 - Опис об’єкту Planet

Поле	Тип	Призначення
planetName	string	Назва планети
surface	SpriteView	Зображення поверхні планети
sky	SpriteView	Зображення неба над планетою
icon	SpriteView	Іконка планети
g	float	Коефіцієнт прискорення вільного падіння
wins	int	Кількість перемог у грі, необхідна, щоб відкрити планету

Таблиця 3.9 - Опис об’єкту Spaceship

Поле	Тип	Призначення
spaceshipName	string	Назва космічного човна
sprite	SpriteView	Зображення космічного човна
mass	float	Маса космічного човна
wins	int	Кількість перемог у грі, необхідна, щоб відкрити космічний човен

Таблиця 3.10 - Опис об’єкту SpaceInfo

Поле	Тип	Призначення
minForce	int	Мінімальна сила, що може бути прикладена до космічного човна
maxForce	int	Максимальна сила, що може бути прикладена до космічного човна
minTimeForce	int	Мінімальна тривалість прикладання сили до космічного човна
maxTimeForce	int	Максимальна тривалість прикладання сили до космічного човна
minAngle	int	Мінімальний кут на початку польоту
maxAngle	int	Максимальний кут на початку польоту
touchdownErrorLimits	Vector2	Допустима похибка відхилення точки приземлення космічного човна при визначенні програшу чи виграшу

3.2.3 Сторонні плагіни

Json .Net Unity робить можливою Json серіалізацію у Unity. Даний плагін зберігає оригінальні простори імен та структуру бібліотеки Newtonsoft Json.Net з підтримкою функцій Unity.

У проекті плагін використовується для збереження прогресу гравця на його пристрої у вигляді Json.

```
using System;
using Newtonsoft.Json;

public class Progress
{
    [JsonProperty("Space")]
    public Points space;

    [JsonProperty("Plane")]
    public Points plane;
}

public class Points
{
    [JsonProperty("wins")]
    public int wins;

    [JsonProperty("looses")]
    public int looses;

    [JsonConstructor]
    public Points(int wins, int looses)
    {
        this.wins = wins;
        this.looses = looses;
    }
}
```

Рисунок 3.8 - Клас прогресу

Бібліотека Newtonsoft.Json дозволяє десеріалізувати дані за допомогою функції JsonConvert.DeserializeObject та серіалізувати дані за допомогою функції JsonConvert.SerializeObject.

```
public class LocalDataController:ILocalDataController
{
    public string Serialize<T>(T jsonObject)
    {
        string json = JsonConvert.SerializeObject(jsonObject);
        return json;
    }

    public T Deserialize<T>(string data)
    {
        T getObject = JsonConvert.DeserializeObject<T>(data);
        return getObject;
    }
```

Рисунок 3.9 - Приклад використання бібліотеки Newtonsoft.Json

Zenject Unity – це фреймворк для впровадження залежностей, розроблений спеціально для Unity. З його допомогою можна розділити проект на слабко пов’язані частини з чіткою сегментацією обов’язків.

Фреймворк має наступні особливості:

– можливість ін'єкції звичайних класів C# та нащадків MonoBehaviour, окремих полів та властивостей;

– підтримання створення об'єктів після ініціалізації з використанням фабрик;

– ін’єкція між різними сценами Unity для передачі даних від однієї сцени до іншої та глобальні ін’єкції;

– підтримання негайного створення об'єктів у момент байндингу та відкладеного створення;

– використання патерну Singleton для створення об’єктів у єдиному екземплярі.

Рисунок 3.10 - Застосування фреймворку Zenject Unity в редакторі

У проекті фреймворк застосовано для прив’язки конкретного класу до інтерфейсу, використання об’єктів у єдиному екземплярі, ін'єкції окремих полів та створення фабрики контролерів.

```
public class HomeInstaller : MonoInstaller<HomeInstaller>, IControllerFactory
{
    [SerializeField]
    private List<Window> _windows;

    public override void InstallBindings()
    {
        Container.BindInstance(_windows);
        Container.Bind<IViewPool>().To<ViewPool>().AsSingle().WithArguments(_windows);

        BindControllers();

        Container.Bind(typeof(IControllerFactory)).FromInstance(this).AsSingle();

        Container.Bind<HomeLoader>().AsSingle().NonLazy();

    }

    public  IController CreateController(WindowType type)
    {
        switch (type)
        {
            case WindowType.HomeWindow:
                return Container.Resolve<HomeController>();

            case WindowType.ProfileWindow:
                return Container.Resolve<ProfileController>();

            case WindowType.RocketsWindow:
                return Container.Resolve<AvailableRocketsController>();

            case WindowType.PlanetsWindow:
                return Container.Resolve<AvailablePlanetsController>();

            case WindowType.MaterialsWindow:
                return Container.Resolve<AvailablePlanesController>();

            default:
                return null;
        }
    }
```

Рисунок 3.11 - Приклад використання фреймворку в коді

3.3 Конструювання програмного забезпечення

3.3.1 UI частина

UI частину складають класи, що відповідають за візуальне наповнення додатку. Усі вони є нащадками класу MonoBehaviour та пов’язані з префабами.

Далі буде наведений короткий опис таких класів.

Таблиця 3.11 - Опис класів нащадків MonoBehaviour

Клас	Опис	Основні UI поля	Основні методи
WindowView	Базовий клас для усіх екранів та спливаючих вікон	-	-
AvailableItemsView	Клас, пов’язаний з екраном показу доступних у грі об’єктів	Грід об’єктів_grid, скрол об’єктів ScrollRect _scroll, кнопка закриття вікна Button _closeButton	SetItems - розміщення отриманих об’єктів у грід та скрол
ProfileView	Клас, пов’язаний з екраном показу прогресу гравця (його програшів та виграшів)	Прогрес у грі про тіло, кинуте під кутом до горизонту ProgressViewData _spaceProgress, прогрес у грі про рух похилою площиною ProgressViewData _planeProgress, кнопка закриття вікна Button _closeButton	SetSpaceProgress, SetPlaneProgress - відтворення виграшів та програшів у іграх

Продовження таблиці 3.11

Retry Window	Клас, пов’язаний з вікном-повідомлення про програш/виграш	Текст вікна TMP_Text _text, кнопка повтору гри Button _retryButton, кнопка віходу в головне меню _closeButton	ShowText - відтворення тексту про виграш/програш
Home View	Клас, пов’язаний з головним екраном	Кнопки відкриття екранів показу доступних у грі об’єктів Button _rocketsButton, _materialsButton, _profileButton, _rocketGameButton, кнопки запуску ігор _planetsGameButton, _planeGameButton	-
BasePrefab	Базовий клас для префабів, копії яких створюються під час гри.	-	-

Продовження таблиці 3.11

Plane View	Клас, пов’язаний з екраном гри про рух похилою площиною	Слайдер зміни маси тіла SliderView _massSlider, слайдер зміни кута площини RotationSliderView _angleSlider, назва матеріалу TMP_Text _materialText, коефицієнт тертя TMP_Text _muText, довжина площини TMP_Text _lengthText; очікувана та поточна швидкості TMP_Text _neededSpeedText, _currentSpeedText, кнопка початку руху Button _moveButton, префаб площини PlanePrefab _plane	Initialize - виставлення усіх потрібних параметрів та текстів, ReturnValues - повернення поточних показників слайдерів, SetObjectPosition - зміна позиції об’єкта, що рухається планкою, SetCurrentSpeed - виставлення поточної швидкості об’єкта, StartMovementAnimation - увімкнення анімації руху об’єкту
AvailableItem	Клас, пов’язаний з префабом доступного у грі об’єкта	Зображення об’єкта Image _icon, зображення недоступности об’єкта GameObject _lockMask, информация про об’єкт TMP_Text _info, кількість перемог для доступу до об’єкту TMP_Text _level	SetProperties - встановлення параметрів об’єкту, ChangeOpened - змінити доступність об’єкту

Продовження таблиці 3.11

Space View	Клас, пов'язаний з екраном гри про тіло, кинуте під кутом до горизонту	Зображення неба та поверхні планети Image _background, _surface, слайдер зміни кута польоту RotationSliderView _angleSlider, слайдер зміни сили SliderView _forceSlider, префаб ракети RocketView _rocket, префаб відстані DestinationView _destination, назва планети та коефіцієнт прискорення вільного падіння TMP_Text _planetName та _g, кнопка початку польоту Button _flyButton, тривалість прикладання сили TMP_Text _influnceDuration, маса ракети TMP_Text _mass	ReturnValues- повернення поточних показників слайдерів, Initialize - виставлення всіх потрібних параметрів та текстів, SetRocketPosition - зміна положення ракети, StartMovementAnimation - увімкнення анімації руху ракети
Destination View	Клас, пов'язаний з префабом відстані від ракети до необхідної точки приземлення	Стрілка позначення відстані RectTransform _arrow, текст відстані TMP_Text _distanceText	Initialize - виставлення всіх потрібних параметрів та текстів

Продовження таблиці 3.11

Plane Prefab	Клас, пов’язаний з префабом площини	Об’єкт, що рухається площиною RectTransform _movingObject, площина RectTransform _plane, зображення матеріалу площини Image _planeMaterial, система партиклів ефекту руху об’єкту UIParticleSystem _particleSystem;	Initialize - виставлення всіх потрібних параметрів та текстів, SetPosition - зміна позиції об’єкта, що рухається планкою, StartMovementAnimation - увімкнення анімації руху об’єкту
Rocket View	Клас, пов’язаний з префабом ракети	Зображення ракети Image _image, система партиклів ефекту руху об’єкту UIParticleSystem _particleSystem	Initialize - виставлення всіх потрібних параметрів, SetPosition - зміна позиції об’єкта, StartMovementAnimation - увімкнення анімації руху об’єкту
Rotation Slider View	Клас, пов’язаний зі слайдером зміни повороту об’єкту, нащадок SliderView	Слайдер Slider _slider, мінімальне, максимальне та поточне значення слайдера TMP_Text _minText, _maxText, _currentText	Initialize - виставлення всіх потрібних параметрів, GetValue - отримання значення слайдера, AddObjectToRotate - додати об’єкт для повороту, ChangeCurrentValue - зміна значення слайдера та поворот об’єктів

Продовження таблиці 3.11

SliderView	Клас, пов’язаний зі слайдером зміни параметрів об’єкту	Слайдер Slider _slider, мінімальне, максимальне та поточне значення слайдера TMP_Text _minText, _maxText, _current Text	Initialize - виставлення всіх потрібних параметрів, GetValue - отримання значення слайдера, ChangeCurrentValue - зміна значення слайдера

3.3.2 Сервісна частина

Класи даної категорії є класичними C# класами.

Спершу розглянемо класи-контролери, що безпосередньо взаємодіють з нащадками MonoBehaviour. Усі вони реалізують інтерфейс IController з єдиним методом Open, що приймає в якості параметра словник Dictionary<string, object> parameters, заповнений даними для ініціалізації.

Таблиця 3.12 - Опис класів контролерів

Клас	Опис та функціонал	Основні методи
Available Planes Controller	Визначення доступних та недоступних гравцеві матеріалів площини, відтворення даних у UI сегменті	InstantiateItems - задання параметрів усіх наявних матеріалів, створення копій префабу AvailableItem, Open - зміна доступності матеріалів, Close - згортання вікна

Продовження таблиці 3.12

Available Planets Controller	Визначення доступних та недоступних гравцеві планет, відтворення даних у UI сегменті	InstantiateItems - задання параметрів усіх наявних планет, створення копій префабу AvailableItem, Open - зміна доступності планет, Close - згортання вікна
Available Rockets Controller	Визначення доступних та недоступних гравцеві ракет та космічних човнів, відтворення даних у UI сегменті	InstantiateItems - задння параметрів усих наявних ракет, створення копій префабу AvailableItem, Open - зміна доступності ракет, Close - згортання вікна
Home Controller	Відкриття екранів доступних елементів та завантаження сцен ігор	OpenWindow - відкриття вікна за типом, OpenScene - завантаження сцени за типом
Plane Controller	Обслуговування гри про рух похилою площиною, забезпечення зв’язку між UI та фізичними обрахунками	ImitateMove - імітація руху площиною за фізичними формулами, OpenRetryWindow - відкриття вікна з результатом гри, CheckSpeedIndent - порівняння результату вирішення задачі з правильною відповіддю, RandomizeValues - рандомізація параметрів задачі, CountSpeed - програмне вирішення задачі

Продовження таблиці 3.12

Plane Retry Window Controller	Відтворення результатів гри про рух похилою площиною у UI сегменті	Open - виведення даних про програш/виграш, ReloadScene - перезавантаження гри, LoadMainScene - перехід на сцену головного меню
Profile Controller	Відтворення даних про прогрес гравця у UI сегменті	Open - відтворення погресу, Close - згортання вікна
Space Controller	Обслуговування гри про рух тіла, кинутого під кутом до горизонту, забезпечення зв’язку між UI та фізичними обрахунками	ImitateFlight - імітація руху тіла, кинутого під кутом до горизонту, за фізичними формулами, OpenRetryWindow - відкриття вікна з результатом гри, CheckDistanceIndent - порівняння результату вирішення задачі з правильною відповіддю, RandomizeValues - рандомізація параметрів задачі, CountDestination - програмне вирішення задачі
Space Retry Window Controller	Відтворення результатів гри прорух тіла, кинутого під кутом до горизонту, у UI сегменті	Open - виведення даних про програш/виграш, ReloadScene - перезавантаження гри, LoadMainScene - перехід на сцену головного меню

Для отримання початкових даних ігор використовуються так звані класи-холдери. За допомогою Zenject вони байндяться як одинаки та доступні з будь-якої сцени. Такі класи підвантажують потрібні дані з ресурсів при першому зверненні та зберігають їх протягом усієї гри.

Таблиця 3.13 - Опис класів холдерів

Клас	Опис та функціонал	Основні методи
GameData Holder	Збереження загальноігрових даних, наприклад, ігрових текстів	GetText - отримання тексту за типом
PlaneData Holder	Збереження даних гри про рух похилою площиною: основних параметрів та списку матеріалів площини	GetRandomMaterial - отримання рандомного матеріалу з доступних, GetMaterial - отримання матеріалу за типом, GetMaterials - отримання усіх матеріалів, GetPlaneInfo - отримання основних параметрів
SpaceData Holder	Збереження даних гри про рух тіла, кинутого під кутом до горизонту: основних параметрів та списків планет і ракет	GetRandomPlanet - отримання рандомної планети з доступних, GetPlanet - отримання планети за типом, GetPlanets - отримання усіх планет, GetRandomSpaceship - отримання рандомної ракети з доступних, GetSpaceship - отримання ракети за типом, GetSpaceships - отримання усіх ракет, GetSpaceInfo - отримання основних параметрів

Продовження таблиці 3.13

Prefabs Holder	Збереження списку префабів для копіювання	GetPrefab - отримання префабу за типом класу
Scales Holder	Збереження даних про масштабування в залежності від розширення екрану	CountRatio - визначення відношення висоти екрану до ширини, public PlaneScale GetPlaneScale, GetRocketScale - отримання масштабів ігор

Крім вище перерахованих класів у проекті присутні класи фізичних обрахунків, збереження прогресу та пул екранів і вікон.

Таблиця 3.14 - Опис класів сервісної частини

Клас	Опис та функціонал	Основні методи
ForceImpulse Physics	Обрахунок імпульсу сили	GetSpeedDiffByForce - обрахунок зміни швидкості за силою та тривалістю її прикладання
InclinedPlane Physics	Обрахунки фізики руху тіла похилою площиною	GetDistance - обрахунок відстані в залежності від часу, GetTime - обрахунок часу руху в залежності від відстані, GetAcceleration - обрахунок прискорення, GetSpeed - обрахунок кінцевої швидкості
ThrownBody Physics	Обрахунки фізики руху тіла, кинутого під кутом до горизонту	GetLocationByTime - обрахунок положення, GetRotationByTime - обрахунок повороту, GetFlightLength - обрахунок дальності польоту

Продовження таблиці 3.14

Progress Handler	Управління даними щодо прогресу гравця	FillBaseProgress - заповнення початкового прогресу, Save - збереження прогресу, AddSpacePoints, AddPlanePoints - нарахування програшу/виграшу у іграх
ViewPool	Створення нових вікон за потреби або увимкнення їх, якщо вікна вже створені, вимкнення вікон	GetController - отримання контролеру з Zenject фабрики за типом, GetView - отримання та увімкнення View за типом, Push - вимкнення View

3.4 Висновки до розділу

У розділі було подано алгоритм дій користувача програмного забезпечення та реакція на них програми. Представлено перелік архітектурних рішень. Також детально описано реалізацію застосунку.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Під час встановлення додатку та протягом його експлуатації можуть виникати помилки, як критичні, що повністю припиняють роботу програми, так і незначні, які, однак, погіршують враження від застосунку. Існують дефекти, які можуть з’явитися ще на стадіях проектування та розробки програмного забезпечення. Інші баги, навпаки, виникають лише при введенні застосунку в експлуатацію.

Виділяють дві найбільш поширені причини виникнення дефектів:

- через недбалість програміста;
- через невідповідність вимог експлуатації поданим у технічному завданні.

Баги у процесі розробки сповільнюють команду та вимагають додаткових ресурсів на вирішення проблем. Дефекти, що з'являються, коли додаток вже запущено, можуть призвести до втрати коштів та клієнтів замовником, а також погіршення репутації виконавця.

Контролювати якість програмного забезпечення потрібно з моменту початку роботи над ним. Оскільки тренажер для вивчення взаємодії фізичних сил є грою, був використаний підхід мануального тестування.

Аби стратегія тестування була максимально ефективною, варто заздалегідь написати тест-план. Стандарт IEEE 829 дозволяє окреслити стратегію тестування, а також зважити ризики, що можуть виникнути в процесі розробки.

4.2 Опис процесів тестування

Далі буде наведений тест-план із детальним описом процесів тестування.

4.2.1 Ідентифікатор тест-плану

Physical Forces Android Mobile Game Testing TP_1.0

4.2.2 Вступ до тест-плану

Цей тест план стосується мобільного тренажеру для вивчення взаємодії фізичних сил, написаного за допомогою ігрового рушія Unity на мові C#.

Окремо у проекті мають бути перевірені:

- сцена головного меню з посиланнями на інші сцени та вікна;
- сцена гри про рух тіла, кинутого під кутом до горизонту;
- сцена гри про рух тіла похилою площиною;
- вікна з переліками доступних об’єктів;
- вікно профілю.

4.2.3 Об’єкт тестування

Об’єктом тестування є гра на мобільний телефон, що має бути запущена на платформі Android. Гра має бути протестована на пристроях з версією Android вище 5ї, включаючи Samsung, Huawei та Xiaomi.

Гра створена для екрану з горизонтальною орієнтацією та розширенням 1920x1080. Гра має бути протестована в горизонтальній орієнтації, але на пристроях з різним розширенням.

4.2.4 Компоненти, що тестуються

Мають бути протестовані функціональні та інтерфейсні особливості ПЗ.

UI компоненти, що мають бути протестовані, включають:

- коректну реакцію клавіш на натискання;
- виведення потрібних екранів та вікон;
- коректне відображення даних у текстових полях;
- правильну реакція слайдерів на дії користувача.

Функціональні компоненти, що мають бути протестовані, включають:

- вхід в гру;
- вихід з гри;
- перехід на сцени ігор;
- збереження прогресу;
- відкривання нових параметрів ігор зі збільшенням виграшів.

Компоненти, що мають бути протестовані у грі про рух тіла, кинутого під кутом до горизонту:

- перезавантаження рівня;
- вихід в меню;
- програш чи перемога;
- можливість вибору кута та сили, прикладеної до об'єкта;
- рандомне задання параметрів задачі;
- слідування об’єктом траєкторії польоту.

Компоненти, що мають бути протестовані у грі про рух тілапохилою площиною:

- перезавантаження рівня;
- вихід в меню;
- програш чи перемога;
- можливість вибору кута площини та маси об'єкта;
- рандомне задання параметрів задачі;
- слідування зміни швидкості об’єкта фізичним формулам.

4.2.5 Підхід

Тести будуть проходити згідно з описаними раніше тест кейсами, збереженими у Qase. Тест кейси буде розділено на 2 групи: функціональні та ті, що стосуються користувацького інтерфейсу. Кожен тест кейс буде помічено як такий, що пройшов або провалився. Для тестів, що провалилися, буде створено окремий репорт на дошці Trello. Репорт міститиме ім’я багу, короткий опис,

серйозність та пріоритетність, кроки для відтворення та скриншот за необхідності.

У процесі створення програмного забезпечення буде використовуватись Debug log від Unity – можливість виводити дані під час компіляції та роботи програми у консоль у редакторі, а також відслідковувати помилки.

Потрібно провести інтеграційні тести, тестуючи окремо кожну гру та головне меню, а потім – увесь додаток в цілому.

Правильність використання фізичних формул буде перевірено за допомогою розв’язання задач мануально.

4.2.6 Критерії проходження тестів

Увесь основний функціонал системи повинен функціонувати, як очікувалося, та окреслено в окремих тестових кейсах. Не має бути критичних дефектів, і допускається 5 або менше не критичних дефектів (наприклад, текст трохи зсунуто відносно кнопки). 100% тестів на функціонал мають пройти. У консолі Unity не повинно з’являтися помилок.

4.2.7 Критерії призупинення та продовження тестування

Тестування має бути зупинено негайно, якщо в консолі Unity з’являється повідомлення про помилку.

4.2.8 Результати проведення тестування

Після тестування, результати буде збережено у Qase. Дефекти будуть описані на Trello в секції ToDo, поки вони не будуть вирішені.

4.2.9 Задачі для проведення тестування

Наступні задачі мають бути виконані:

- підготовано план тестування;
- визначено та описано функціональні характеристики додатку;

– підготовано середовище для тестування (створено інструкцію користувача, знайдено сторонній фізичний калькулятор для перевірки);

– виконано тести;

– підготовано фінальний звіт.

4.2.10Технічні потреби

Гра має бути протестована на пристроях з версією Android вище 5ї.

4.2.11Обов’язки

Оскільки тренажер є інді-грою, тестуванням та розробкою займається одна людина. Вона несе відповідальність за всі ризики та слідування плану тестування. Основні завдання тестувальника – створити так багато тест кейсів, як можливо, та описати дефекти.

4.2.12Необхідні компетенції та тренінги

Тестувальнику необхідно мати хороші знання з Unity, C#, а також добре розумітися на правилах ігор. Важливим критерієм є знання шкільної програми з фізики.

4.2.13Розклад

Написання тест-плану має відбутись до початку роботи над проектом, але після визначення вимог до продукту.

Тестування має розпочатись паралельно розробці.

Фінальне коло тестування має відбутись після закінчення роботи над основними модулями продукту та тривати не менше, ніж 1 тиждень.

4.2.14Ризики

Таблиця 4.1 - Ризики

Ризик	Засіб протидії
Зміна вимог	Забезпечити гнучкість ПЗ для легкого внесення незначних змін з ранніх етапів розробки.
Непідготовленість модулів проекту до тестування.	Пришвидшити розробку.
Пошкодження даних Unity.	Використання засобів контролю версій для повернення до останніх збережених даних.

4.3 Висновки до розділу

У даному розділі було описано тест план, згідно з якім має тестуватись програмне забезпечення. Наведено компоненти, що мають бути протестовані, обов’язки тестувальника та можливі ризики.

5 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання даного застосунку під платформу Android спершу потрібно створити збірку. Це має відбуватись наступним чином:

– Крок 1. У редакторі Unity перейти на платформу Android. Підвантажити та встановити Android SDK та Android NDK.

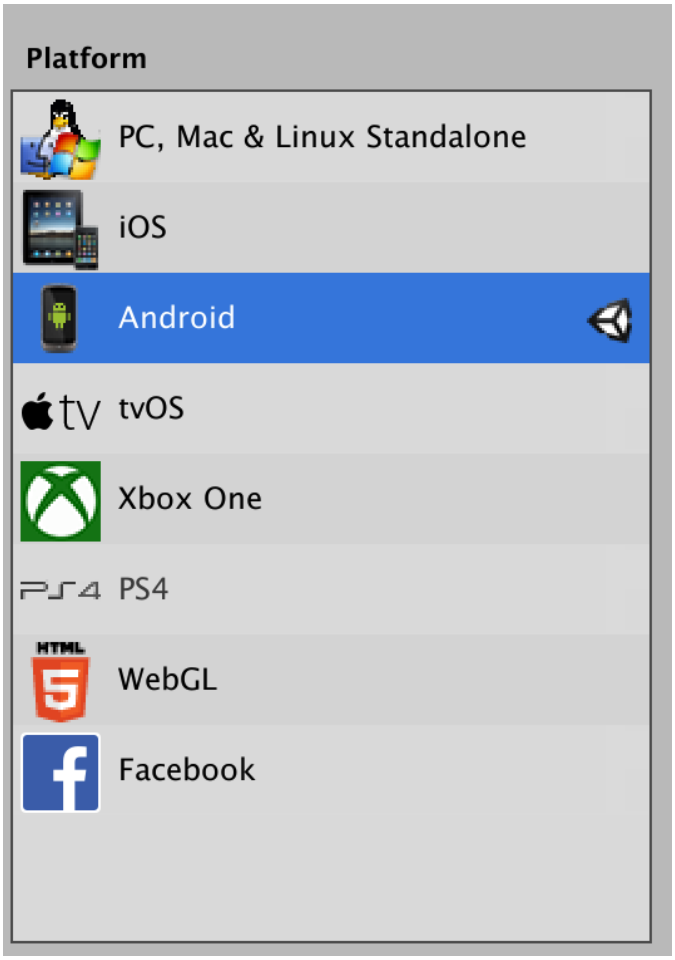


Рисунок 5.1 – Вибір платформи Android у Unity

– Крок 2. У вкладці ігрових налаштувань зазначити назву та версію збірки, визначити необхідні дозволи, орієнтацію, іконку, тощо.

– Крок 3. Зібрати проект у файл розширення .apk.

Аби встановити програму на девайс, необхідно виконати наступні кроки:

– Крок 1. Завантажити збірку у вигляді .apk файлу.
– Крок 2. Надати дозвіл на встановлення додатку.
– Крок 3. Дочекатися завершення установки та відкрити додаток.

5.2 Робота з програмним забезпеченням

Опис процесу роботи з програмним забезпеченням представлено у вигляді інструкції користувача, яка наведена у документі КПІ.ІП-6115.045490.06.34 «МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY. Керівництво користувача».

5.3 Висновки до розділу

У даному розділі наведено алгоритми створення збірки та її встановлення на пристрій. Також подано ілюстровану інструкцію користувача.

ВИСНОВКИ

В рамках дипломного проекту розроблений мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity.

Тема дипломного проекту є актуальною, оскільки присвячена розробці ігрового застосування для посилення інтересу школярів до вивчення фізики. Розробка може бути використана як один із засобів дистанційного навчання. Також програмне забезпечення розширює існуючі ігрові механіки, маючи функціонал зміни коефіцієнту вільного падіння як параметра для імітації руху тіла, кинутого під кутом до горизонту.

У розділі «Аналіз вимог до програмного забезпечення» було оглянуто предметну область, а саме фізичні процеси, що імітуються у грі. Крім того, було досліджено наявні на ринку програми-аналоги на предмет переваг та недоліків. Спираючись на проаналізовану інформацію, було сформовано функціональні вимоги до програмного забезпечення, вимоги до UI-складової, а також поставлено цілі, що мають бути досягнуті під час розробки.

У розділі «Вибір технологій програмування» було описано використані під час розробки мову програмування та ігровий рушій. Також проаналізовано переваги даних технологій, що роблять їх зручними для розробки програмного забезпечення.

У розділі «Моделювання та конструювання програмного забезпечення» було подано алгоритм дій користувача та реакцію програми на них. Було обрано та проаналізовано архітектуру програми, структуру даних, що формують параметри задач, допоміжні фреймворки, застосовані у процесі розробки. Окрім цього, було проведено конструювання програмного забезпечення та подано опис основних класів програми.

У розділі «Аналіз якості та тестування програмного забезпечення» було подано тест-план, що має бути застосований до створеного програмного забезпечення.

У розділі «Впровадження та супровід програмного забезпечення» було описано кроки, що має здійснити програміст задля отримання файлу встановлення додатку та користувач для встановлення додатку на смартфон. Крім того подано ілюстровану інструкцію користувача.

Результатом даного дипломного проекту є мобільний тренажер для вивчення взаємодії фізичних сил, що складається з головного меню, гри про рух тіла, кинутого під кутом до горизонту, та гри про рух тіла похилою площиною.

Таким чином, розроблене програмне забезпечення вирішує всі поставлені перед ним задачі та відповідає поставленим вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1) Ігровий рушій [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://uk.wikipedia.org/wiki/Ігровий_рушій

2) LINQ [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://uk.wikipedia.org/wiki/LINQ

3) JSON [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/JSON

4) Рух тіла, кинутого під кутом до горизонту [Електронний ресурс] (Стаття) / Disted. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://disted.edu.vn.ua/courses/learn/2900

5) Імпульс сили [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. –Режим доступу: https://uk.wikipedia.org/wiki/Імпульс_сили

6) Імпульс тіла [Електронний ресурс] (Стаття) / Repetitor. – Електрон. дан. (1 файл). – 2018. –Режим доступу: https://repetitor.org.ua/impuls-tila-2

7) Божинова Ф. Я. Фізика. 8 клас: Підручник. - Х.: Ранок-НТ, 2008. - 256 с.

8) Unity (игровой движок) [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://ru.wikipedia.org/wiki/Unity(игровой_движок)

9) Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.: Питер, 2013. — 896 с.

10) Model View Controller [Електронний ресурс] (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller

11) Горошко А.М., Чиж О. Й. Універсальний шкільний довідник з фізики. 7-11 класи.– Тернопіль : Підручники і посібники, 2013.– 223 с.

12) Unity User Manual [Електронний ресурс] (Стаття) / Unity3d. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://docs.unity3d.com/Manual/index.html

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2020 р.

МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

Технічне завдання

КПІ.ІП-6115.045490.03.91

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Л.С.Кухарець

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity

Галузь застосування:

Наведене технічне завдання поширюється на розробку гри під платформу Android «Мобільний тренажер для вивчення взаємодії фізичних сил», котра використовується для візулізації вирішення задач на взаємодію фізичних сил та призначена для вивчення фізики шкільної програми.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Мобільного тренажеру для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут» (НТУУ «КПІ»).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для вирішення задач на взаємодію фізичних сил у формі гри.

Метою даної розробки є посилення інтересу школярів до вивчення фізики, їх мотивування правильно вирішувати задачі, а також розширення асортименту існуючих засобів дистанційного навчання.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

- при вході у гру завантаження головного екрану з відображенням іконок міні-ігор;
- наявність екранів перегляду доступних параметрів ігрового середовища у вигляді скролу;
- перегляд програшів та перемог для кожної міні-гри у вікні профілю;
- наявність міні-гри про рух тіла, кинутого під кутом до горизонту;
- наявність міні-гри про рух тіла похилою площиною;
- генгерація рандомних параметрів задачі при завантаженні сцени гри;
- відтворення руху тіла за фізичними законами після натискання клавіші старту;
- збереження програшу чи виграшу після припинення руху тіла.

4.1.2Розробку виконати на платформі Windows 10 або macOS Catalina.

4.2 Вимоги до надійності

4.2.1Передбачити контроль введення інформації за допомогою слайдерів та кнопок користувацького інтерфейсу.

4.2.2Передбачити захист від некоректних дій користувача шляхом блокування кнопок після старту руху об’єктів та обмеження значення слайдерів мінімально та максимально допустимими значеннями.

4.2.3Забезпечити збереження прогресу при помилках, які пов’язані з програмним забезпеченням, за допомогою збереження інформації у Json файлі після кожного її оновлення.

4.3 Умови експлуатації

4.3.1Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2Обслуговування

Вимоги до обслуговування не пред‘являються.

4.3.3Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не пред‘являються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1Програмне забезпечення повинно функціонувати на мабільних телефонах та планшетах з системою Android..

4.4.2Мінімальна конфігурація технічних засобів:

Система пристрою: Android 5.0 “Lollipop” і вище.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1Програмне забезпечення повинно працювати під управлінням системи Android.

4.5.2Вхідні дані повинні бути представлені в наступному форматі: скриптовані об’єкти, взаємодія зі слайдерами та кнопками користувацького інтерфейсу.

4.5.3Результати повинні бути представлені в наступному форматі: Json файл зі збереженням прогресу гравця, візуальний інтерфейс.

4.5.4Програмне забезпечення повинно бути створеним за допомогою ігрового рушія Unity на мові C#..

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наведені нижче документи.

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування.

5.4 Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна класів програмного забезпечення

5.4.2 Схема структурна варіантів використань

5.4.3 Схема структурна класів програмного забезпечення

5.4.4 Схема структурна діаграма діяльності користувача та програми

5.4.5 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	05.05.2020	
2.	Розробка технічного завдання	10.05.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	15.05.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	20.05.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	26.05.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	30.05.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	03.06.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	03.06.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	05.06.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

Опис програми

КПІ.ІП-6115.045490.04.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Л.С. Кухарець

Київ – 2020 року

Тексти програмного коду

МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

(Наименування програми (документа))

DVD-R

(Вид носія даних)

36 арк, 50 Мб

(Обсяг програми (документа) , арк.,)

Київ - 2020

1 КОД UI ЧАСТИНИ

```
public class AvailableItem:BasePrefab
{
    [SerializeField]
    private Image _icon;

    [SerializeField]
    private GameObject _lockMask;

    [SerializeField]
    private TMP_Text _info;

    [SerializeField]
    private TMP_Text _level;

    [SerializeField]
    private string _infoFormat;

    [SerializeField]
    private string _levelFormat;

    private string _infoText;

    private string _lockedInfoText;

    public void SetProperties(Sprite icon, int level, string info, string
lockedInfo)
    {
        _icon.sprite = icon;
        _level.text = string.Format(_levelFormat, level);
        _infoText = info;
        _lockedInfoText = lockedInfo;
    }

    public void ChangeOpened(bool isOpened)
    {
        _icon.gameObject.SetActive(isOpened);
        _lockMask.SetActive(!isOpened);
        _info.text = string.Format(_infoFormat,
isOpened?_infoText:_lockedInfoText);
    }

}

public class BasePrefab:MonoBehaviour
{

}

public class DestinationView:MonoBehaviour
{
    [SerializeField]
    private RectTransform _arrow;

    [SerializeField]
    private TMP_Text _distanceText;

    [SerializeField]
    private string _distanceTextFormat;
```

```
  public void Initialize(Vector3 startPosition, float distance, float coeff,
float indent)
    {
        _distanceText.text = string.Format(_distanceTextFormat, distance);

        _arrow.offsetMax=new Vector2 (distance* coeff, _arrow.offsetMax.y);
    }
}

public class PlanePrefab:MonoBehaviour
{
    [SerializeField]
    private RectTransform _movingObject;

    [SerializeField]
    private RectTransform _plane;

    [SerializeField]
    private Image _planeMaterial;

    [SerializeField]
    private UIParticleSystem _particleSystem;


    public void Initialize(float length, Sprite material)
    {
        _plane.offsetMax = new Vector2(length, _plane.offsetMax.y);
        _planeMaterial.sprite = material;
    }

    public void SetPosition(Vector3 indent)
    {
        _movingObject.anchoredPosition =  - indent;

    }

    private void SetParticleRotation(float angle)
    {
        float x = Mathf.Cos(angle * Mathf.Deg2Rad);

        float y = Mathf.Sin(angle * Mathf.Deg2Rad);

        _particleSystem.ChangeDirection(new Vector2(x, y));
    }

    public void StartMovementAnimation(float angle)
    {
        SetParticleRotation(angle);
        _particleSystem.Play();
    }

    internal void StopMovementAnimation()
    {
        _particleSystem.Stop();
    }
}

public class QuitView:MonoBehaviour
{
    [SerializeField]
```

```
    private SceneType _type;

    void Update()
    {
        if (Application.platform == RuntimePlatform.Android)
        {
            if (Input.GetKeyDown(KeyCode.Escape))
            {
                if (_type == SceneType.HomeScreen)
                {
                    Application.Quit();
                }
                else
                {
                    SceneLoader.LoadScene(SceneType.HomeScreen);
                }
            }
        }
    }
}

public class RocketView:MonoBehaviour
{

    [SerializeField]
    private Image _image;

    [SerializeField]
    private UIParticleSystem _particleSystem;


    private Vector3 _startPosition;


   public void Initialize(SpaceshipInfo spaceship)
    {
        _startPosition = transform.localPosition;

        _image.sprite = spaceship.sprite.sprite;

        SetPosition(Vector3.zero, 45);
    }


    public void SetPosition(Vector3 indent, float angle)
    {
        transform.localPosition = _startPosition+indent;
        transform.eulerAngles = new Vector3(0, 0, angle);

        SetParticleRotation(angle);
    }

    private void SetParticleRotation(float angle)
    {
      float x = Mathf.Cos(angle * Mathf.Deg2Rad);

        float y= Mathf.Sin(angle * Mathf.Deg2Rad);

        _particleSystem.ChangeDirection(new Vector2(-x, -y));
    }
```

```
    public void StartMovementAnimation()
    {
        _particleSystem.Play();
    }

    internal void StopMovementAnimation()
    {
        _particleSystem.Stop();
    }
}

public class RotationSliderView:SliderView
{
    private List<GameObject> _objectsToRotate=new List<GameObject>();

    public void AddObjectToRotate(GameObject obj)
    {
        _objectsToRotate.Add(obj);
    }

    protected override void ChangeCurrentValue(float sliderValue)
    {
        base.ChangeCurrentValue(sliderValue);

        foreach(var obj in _objectsToRotate)
        {
            obj.transform.eulerAngles = new Vector3(0, 0, current);
        }
    }
}

public class SliderView : MonoBehaviour
{
    [SerializeField]
    private Slider _slider;

    [SerializeField]
    private TMP_Text _minText;

    [SerializeField]
    private TMP_Text _maxText;

    [SerializeField]
    private TMP_Text _currentText;

    private int _min;

    private int _max;

    protected int current;

    public void Initialize(int min, int max)
    {
        _min = min;
        _minText.text = _min.ToString();

        _max = max;
        _maxText.text = _max.ToString();

        _slider.onValueChanged.AddListener((value) => {
ChangeCurrentValue(value); });
```

```
            _slider.value = 0.5f;
        }

        public int GetValue()
        {
            return current;
        }

        protected virtual void ChangeCurrentValue(float sliderValue)
        {
            current = _min + (int)((_max - _min) * sliderValue);
            _currentText.text = current.ToString();
        }
    }

    public class AvailableItemsView : WindowView
    {
        [SerializeField]
        private Transform _grid;

        [SerializeField]
        private ScrollRect _scroll;

        [SerializeField]
        private Button _closeButton;

        public  Action onClose;


        private void Start()
        {
            _closeButton.onClick.AddListener(() => onClose?.Invoke());
        }

        private void OnEnable()
        {
            _scroll.normalizedPosition = new Vector2(0, 0);
        }

        public void SetItems(List<AvailableItem> items)
        {
            foreach(var item in items)
            {
                item.transform.SetParent(_grid);
            }

            _scroll.normalizedPosition = new Vector2(0, 0);
        }
    }

    public class HomeView: WindowView
    {
        [SerializeField]
        private Button _rocketsButton;

        [SerializeField]
        private Button _materialsButton;

        [SerializeField]
        private Button _profileButton;
```

```
    [SerializeField]
    private Button _rocketGameButton;

    [SerializeField]
    private Button _planetsGameButton;

    [SerializeField]
    private Button _planeGameButton;

   public Action<SceneType> onSceneOpen;

    public Action<WindowType> onWindowOpen;

    private void Start()
    {
        _rocketsButton.onClick.AddListener(() =>
onWindowOpen?.Invoke(WindowType.RocketsWindow));

        _planetsGameButton.onClick.AddListener(() =>
onWindowOpen?.Invoke(WindowType.PlanetsWindow));

        _materialsButton.onClick.AddListener(() =>
onWindowOpen?.Invoke(WindowType.MaterialsWindow));

        _profileButton.onClick.AddListener(() =>
onWindowOpen?.Invoke(WindowType.ProfileWindow));

        _rocketGameButton.onClick.AddListener(() =>
onSceneOpen?.Invoke(SceneType.RocketGame));

        _planeGameButton.onClick.AddListener(() =>
onSceneOpen?.Invoke(SceneType.PlaneGame));


    }
}

public class PlaneView:WindowView
{

    [SerializeField]
    private SliderView _massSlider;

    [SerializeField]
    private RotationSliderView _angleSlider;

    [SerializeField]
    private TMP_Text _materialText;

    [SerializeField]
    private string _materialFormat;

    [SerializeField]
    private TMP_Text _muText;

    [SerializeField]
    private string _muFormat;

    [SerializeField]
    private TMP_Text _lengthText;
```

```
    [SerializeField]
    private string _lengthFormat;

    [SerializeField]
    private TMP_Text _neededSpeedText;

    [SerializeField]
    private string _neededSpeedFormat;

    [SerializeField]
    private TMP_Text _currentSpeedText;

    [SerializeField]
    private string _currentSpeedFormat;

    [SerializeField]
    private Button _moveButton;

    [SerializeField]
    private PlanePrefab _plane;

    private bool _isButtonlocked;


    public Action<float, float> onMoveStart;

    private void Start()
    {
        _moveButton.onClick.AddListener(() => ReturnValues());
    }

    public void Initialize(PlaneInfo planeInfo, PlaneMaterial material, float
length)
    {
        _plane.Initialize(length, material.view.sprite);
        _massSlider.Initialize(planeInfo.minMass, planeInfo.maxMass);

        _angleSlider.AddObjectToRotate(_plane.gameObject);
        _angleSlider.Initialize(planeInfo.minAngle, planeInfo.maxAngle);


        _materialText.text = string.Format(_materialFormat,
material.materialName);

        _muText.text = string.Format(_muFormat, material.mu);

        _lengthText.text = string.Format(_lengthFormat, length);

    }

    private void ReturnValues()
    {
        if (!_isButtonlocked)
        {
            _isButtonlocked = true;
            onMoveStart.Invoke(_angleSlider.GetValue(), _massSlider.GetValue());
        }
    }
```

```
    public void SetObjectPosition(Vector3 indent)
    {
        _plane.SetPosition(indent);
    }

    public void SetCurrentSpeed(float speed)
    {
        _currentSpeedText.text = string.Format(_currentSpeedFormat, (int)
speed);
    }

    public void SetFinalSpeed(float speed)
    {
       _neededSpeedText.text = string.Format(_neededSpeedFormat, speed);
    }

    public void StartMovementAnimation()
    {
        _plane.StartMovementAnimation(_angleSlider.GetValue());
    }

    public void StopMovementAnimation()
    {
        _plane.StopMovementAnimation();
    }

}

public class ProfileView:WindowView
{
    [SerializeField]
    private ProgressViewData _spaceProgress;

    [SerializeField]
    private ProgressViewData _planeProgress;

    [SerializeField]
    private string _winFormat;

    [SerializeField]
    private string _looseFormat;

    [SerializeField]
    private Button _closeButton;

    public Action onClose;

    private void Start()
    {
        _closeButton.onClick.AddListener(() => onClose?.Invoke());
    }

    public void SetSpaceProgress(int wins, int looses)
    {
        _spaceProgress.wins.text = string.Format(_winFormat, wins);
        _spaceProgress.looses.text = string.Format(_looseFormat, looses);
    }

    public void SetPlaneProgress(int wins, int looses)
    {
        _planeProgress.wins.text = string.Format(_winFormat, wins);
```

```
            _planeProgress.looses.text = string.Format(_looseFormat, looses);
        }
    }

    [Serializable]
    public class ProgressViewData
    {
        [SerializeField]
        public TMP_Text wins;

        [SerializeField]
        public TMP_Text looses;
    }

    public class RetryWindow:WindowView
    {
        [SerializeField]
        private TMP_Text _text;

        [SerializeField]
        private Button _retryButton;

        [SerializeField]
        private Button _closeButton;

        public Action onRetry;
        public Action onClose;

        public void Start()
        {
            _retryButton.onClick.AddListener(() => onRetry?.Invoke());
            _closeButton.onClick.AddListener(() => onClose?.Invoke());
        }

        public void ShowText(string text)
        {
           _text.text = text;
        }

    }

    public class SpaceView: WindowView
    {
        [SerializeField]
        private Image _background;

        [SerializeField]
        private Image _surface;

        [SerializeField]
        private RotationSliderView _angleSlider;

        [SerializeField]
        private RocketView _rocket;

        [SerializeField]
        private SliderView _forceSlider;

        [SerializeField]
        private DestinationView _destination;
```

```
    [SerializeField]
    private TMP_Text _planetName;

    [SerializeField]
    private TMP_Text _g;

    [SerializeField]
    private string _gFormat;

    [SerializeField]
    private Button _flyButton;

    [SerializeField]
    private TMP_Text _influnceDuration;

    [SerializeField]
    private string _influnceDurationFormat;

    [SerializeField]
    private TMP_Text _mass;

    [SerializeField]
    private string _massFormat;

    private bool _isButtonlocked;

    public Action<int, int> onFlightStart;

    private void Start()
    {
        _flyButton.onClick.AddListener(() => ReturnValues());
    }

    private void ReturnValues()
    {
        if (!_isButtonlocked)
        {
            _isButtonlocked = true;

            onFlightStart.Invoke(_angleSlider.GetValue(),
_forceSlider.GetValue()* 1000000);
        }
    }

    public void Initialize(SpaceInfo spaceInfo, SpaceshipInfo spaceship,
PlanetInfo planet, int forceDuration)
    {
        _rocket.Initialize(spaceship);

        _forceSlider.Initialize(spaceInfo.minForce/1000000,
spaceInfo.maxForce/1000000);

        _angleSlider.Initialize(spaceInfo.minAngle, spaceInfo.maxAngle);

        _angleSlider.AddObjectToRotate(_rocket.gameObject);

        _background.sprite = planet.sky.sprite;

        _surface.sprite = planet.surface.sprite;

        _planetName.text = planet.planetName;
```

```
        _g.text = string.Format(_gFormat, planet.g);

        _mass.text = string.Format(_massFormat, spaceship.mass);

        _influnceDuration.text = string.Format(_influnceDurationFormat,
forceDuration);

    }

    public void SetRocketPosition(Vector3 indent, float angle)
    {
        _rocket.SetPosition(indent, angle);
    }

    public void SetDistance(float distance, float coeff, float indent)
    {
        _destination.Initialize(_rocket.gameObject.transform.localPosition,
distance, coeff, indent);
    }

    public void StartMovementAnimation()
    {
        _rocket.StartMovementAnimation();
    }

    public void StopMovementAnimation()
    {
        _rocket.StopMovementAnimation();
    }


}

public class WindowView:MonoBehaviour
{

}
```

2 КОД СЕРВІСНОЇ ЧАСТИНИ

```
public class AvailablePlanesController : IController, IDisposable
{
    private AvailableItemsView _view;

    private AvailableItem _prefab;

    private IViewPool _windowPool;

    private ProgressHandler _progressHandler;

    private IPlaneDataHolder _planeDataHolder;

    private IGameDataHolder _gameDataHolder;

    private List<AvailableItem> _items= new List<AvailableItem>();

    public AvailablePlanesController(IPrefabsHolder prefabsHoder, IViewPool
windowPool, ProgressHandler progressHandler, IPlaneDataHolder planeDataHolder,
IGameDataHolder gameDataHolder)
    {
        _windowPool = windowPool;
        _progressHandler = progressHandler;
        _planeDataHolder = planeDataHolder;
        _gameDataHolder = gameDataHolder;

        _view = (AvailableItemsView)
_windowPool.GetView(WindowType.MaterialsWindow);

        _view.onClose += Close;

        _prefab = prefabsHoder.GetPrefab<AvailableItem>();

        InstantiateItems();

        _view.SetItems(_items);
    }

    private void InstantiateItems()
    {
        foreach(var material in _planeDataHolder.GetMaterials())
        {
            AvailableItem newItem = GameObject.Instantiate(_prefab);

            string info = "Name: " + material.materialName;

            newItem.SetProperties(material.icon.sprite, material.wins, info,
_gameDataHolder.GetText(TextType.LockedItemInfo));

            _items.Add(newItem);

        }
    }

    public void Open(Dictionary<string, object> parameters = null)
    {
```

```
        _view =
(AvailableItemsView)_windowPool.GetView(WindowType.MaterialsWindow);

        int wins = _progressHandler.progress.plane.wins;

        var materials = _planeDataHolder.GetMaterials();

        for (int i = 0; i < materials.Count; i++)
        {
            bool isOpened = materials[i].wins <= wins;
            _items[i].ChangeOpened(isOpened);
        }
    }

    public void Close()
    {
        _windowPool.Push(WindowType.MaterialsWindow);
    }

    public void Dispose()
    {
        _view.onClose -= Close;
    }
}

public class AvailablePlanetsController : IController, IDisposable
{
    private AvailableItemsView _view;

    private AvailableItem _prefab;

    private IViewPool _windowPool;

    private ProgressHandler _progressHandler;

    private ISpaceDataHolder _spaceDataHolder;

    private IGameDataHolder _gameDataHolder;

    private List<AvailableItem> _items = new List<AvailableItem>();

    public AvailablePlanetsController(IPrefabsHolder prefabsHoder, IViewPool
windowPool, ProgressHandler progressHandler, ISpaceDataHolder spaceDataHolder,
IGameDataHolder gameDataHolder)
    {
        _windowPool = windowPool;
        _progressHandler = progressHandler;
        _spaceDataHolder = spaceDataHolder;
        _gameDataHolder = gameDataHolder;

        _view = _view = (AvailableItemsView)
_windowPool.GetView(WindowType.PlanetsWindow);

        _view.onClose += Close;

        _prefab = prefabsHoder.GetPrefab<AvailableItem>();

        InstantiateItems();

        _view.SetItems(_items);
    }
```

```
    private void InstantiateItems()
    {
        foreach (var planet in _spaceDataHolder.GetPlanets())
        {
            AvailableItem newItem = GameObject.Instantiate(_prefab);

            string info = "Name: " + planet.planetName;

            newItem.SetProperties(planet.icon.sprite, planet.wins, info,
_gameDataHolder.GetText(TextType.LockedItemInfo));

            _items.Add(newItem);

        }
    }

    public void Open(Dictionary<string, object> parameters = null)
    {
        _view = _view =
(AvailableItemsView)_windowPool.GetView(WindowType.PlanetsWindow);

        int wins = _progressHandler.progress.plane.wins;

        var materials = _spaceDataHolder.GetPlanets();

        for (int i = 0; i < materials.Count; i++)
        {
            bool isOpened = materials[i].wins <= wins;
            _items[i].ChangeOpened(isOpened);
        }
    }

    public void Close()
    {
        _windowPool.Push(WindowType.PlanetsWindow);
    }

    public void Dispose()
    {
        _view.onClose -= Close;
    }
}

public class AvailableRocketsController : IController, IDisposable
{
    private AvailableItemsView _view;

    private AvailableItem _prefab;

    private IViewPool _windowPool;

    private ProgressHandler _progressHandler;

    private ISpaceDataHolder _spaceDataHolder;

    private IGameDataHolder _gameDataHolder;

    private List<AvailableItem> _items = new List<AvailableItem>();
```

```
    public AvailableRocketsController(IPrefabsHolder prefabsHoder, IViewPool
windowPool, ProgressHandler progressHandler, ISpaceDataHolder spaceDataHolder,
IGameDataHolder gameDataHolder)
    {
        _windowPool = windowPool;
        _progressHandler = progressHandler;
        _spaceDataHolder = spaceDataHolder;
        _gameDataHolder = gameDataHolder;

        _view = _view =
(AvailableItemsView)_windowPool.GetView(WindowType.RocketsWindow);

        _view.onClose += Close;

        _prefab = prefabsHoder.GetPrefab<AvailableItem>();

        InstantiateItems();

        _view.SetItems(_items);
    }

    private void InstantiateItems()
    {
        foreach (var spaceship in _spaceDataHolder.GetSpaceships())
        {
            AvailableItem newItem = GameObject.Instantiate(_prefab);

            string info = "Name: " + spaceship.spaceshipName;

            newItem.SetProperties(spaceship.sprite.sprite, spaceship.wins, info,
_gameDataHolder.GetText(TextType.LockedItemInfo));

            _items.Add(newItem);
        }
    }

    public void Open(Dictionary<string, object> parameters = null)
    {
        _view = _view =
(AvailableItemsView)_windowPool.GetView(WindowType.RocketsWindow);

        int wins = _progressHandler.progress.plane.wins;

        var materials = _spaceDataHolder.GetSpaceships();

        for (int i = 0; i < materials.Count; i++)
        {
            bool isOpened = materials[i].wins <= wins;
            _items[i].ChangeOpened(isOpened);
        }
    }

    public void Close()
    {
        _windowPool.Push(WindowType.RocketsWindow);
    }

    public void Dispose()
    {
        _view.onClose -= Close;
    }
```

```
}

public class HomeController:IController, IDisposable
{
    private HomeView _view;

    private IViewPool _windowPool;

    public HomeController(IViewPool windowPool)
    {
        _windowPool = windowPool;

        _view = (HomeView)_windowPool.GetView(WindowType.HomeWindow);

        _view.onSceneOpen += OpenScene;
        _view.onWindowOpen += OpenWindow;
    }

    private void OpenWindow(WindowType type)
    {
        IController controller = _windowPool.GetController(type);
        controller.Open();
    }

    private void OpenScene(SceneType type)
    {
        SceneLoader.LoadScene(type);
    }

    public void Open(Dictionary<string, object> parameters = null)
    {

    }

    public void Dispose()
    {
        _view.onSceneOpen -= OpenScene;

        _view.onWindowOpen -= OpenWindow;
    }
}

public interface IController
{
    void Open(Dictionary<string, object> parameters=null);
}

public class PlaneController: IController, IDisposable
{
    private PlaneView _view;

    private PlaneMaterial _material;

    private IViewPool _windowPool;

    private float _planeLength;

    private AsyncProcessor _asyncProcessor;

    private IInclinedPlanePhysics _basePhysics;
```

```
    private ProgressHandler _progressHandler;

    private IPlaneDataHolder _planeDataHolder;

    private PlaneInfo _planeInfo;

    private IScalesHolder _scalesHandler;

    private float _speed;

    private float _time = 0;

    public PlaneController(AsyncProcessor asyncProcessor, IViewPool windowPool,
ProgressHandler progressHandler, IPlaneDataHolder planeDataHolder,
IInclinedPlanePhysics physics, IScalesHolder scalesHandler)
    {
        _windowPool = windowPool;
        _asyncProcessor = asyncProcessor;
        _progressHandler = progressHandler;
        _planeDataHolder = planeDataHolder;
        _scalesHandler=scalesHandler;
        _basePhysics = physics;

        _view = _view =
(PlaneView)_windowPool.GetView(WindowType.PlaneMainWindow);

        _view.onMoveStart += ImitateMove;

        _planeInfo = _planeDataHolder.GetPlaneInfo();

        RandomizeValues();

        CountSpeed();
    }

    private void ImitateMove(float angle, float mass)
    {
        _view.StartMovementAnimation();
        _asyncProcessor.StartCoroutine(MoveCoroutine(angle, mass));
    }

    private IEnumerator MoveCoroutine(float angle, float mass)
    {
        float distance;
        float a = _basePhysics.GetAcceleration(_planeInfo.g, angle,
_material.mu, mass);
        float finalSpeed = _basePhysics.GetSpeed(a,
_basePhysics.GetTime(_planeLength, a), 0);
        do
        {
            _time += Time.deltaTime*5;

            distance = _basePhysics.GetDistance(a, _time);

            Vector3 indent = new Vector3(distance, 0);

            float speed = _basePhysics.GetSpeed(a, _time, 0);

            _view.SetObjectPosition(indent);

            _view.SetCurrentSpeed(speed);
```

```
            yield return null;

        } while (distance < _planeLength);

        bool isWin = CheckSpeedIndent(finalSpeed);
        _progressHandler.AddPlanePoints(isWin ? 1 : -1);
        _view.StopMovementAnimation();
        OpenRetryWindow(isWin);
    }

    private void OpenRetryWindow(bool isWinner)
    {
        IController retryController =
_windowPool.GetController(WindowType.PlaneRetryWindow);

        Dictionary<string, object> parameters = new Dictionary<string,
object>();

        parameters.Add("GameResult", isWinner);

        retryController.Open(parameters);
    }

    private bool CheckSpeedIndent(float currentFinalSpeed)
    {
        return currentFinalSpeed >= _speed - _planeInfo.finalSpeedLimits.x &&
            currentFinalSpeed <= _speed + _planeInfo.finalSpeedLimits.y;
    }

    private void RandomizeValues()
    {
        _material =
_planeDataHolder.GetRandomMaterial(_progressHandler.progress.plane.wins);
        _planeLength = UnityEngine.Random.Range(_planeInfo.minLength,
_planeInfo.maxLength+1);
        _view.Initialize(_planeInfo, _material, _planeLength);
    }

    private void CountSpeed()
    {
        float angle = UnityEngine.Random.Range(_planeInfo.minAngle,
_planeInfo.maxAngle + 1);
        float mass =  UnityEngine.Random.Range(_planeInfo.minMass,
_planeInfo.maxMass + 1);

        float a = _basePhysics.GetAcceleration(_planeInfo.g, angle,
_material.mu, mass);
      //  Debug.Log("time: " + _basePhysics.GetTime(20, a));
        _speed = _basePhysics.GetSpeed(a, _basePhysics.GetTime(_planeLength, a),
0);

        _view.SetFinalSpeed(_speed);
    }

    public void Open(Dictionary<string, object> parameters = null)
    {

    }

    public void Dispose()
```

```
        {
            _view.onMoveStart += ImitateMove;
        }
    }

    public class PlaneRetryWindowController : IController, IDisposable
    {
        private RetryWindow _view;

        private IViewPool _windowPool;

        private IGameDataHolder _gameDtaHolder;

        public PlaneRetryWindowController(IViewPool windowPool, IGameDataHolder
    gameDtaHolder)
        {
            _windowPool = windowPool;
            _gameDtaHolder = gameDtaHolder;

            _view = (RetryWindow) _windowPool.GetView(WindowType.PlaneRetryWindow);

            _view.onClose += LoadMainScene;
            _view.onRetry += ReloadScene;
        }

        private void ReloadScene()
        {
            _windowPool.Push(WindowType.PlaneRetryWindow);
            SceneLoader.LoadScene(SceneType.PlaneGame);
        }

        private void LoadMainScene()
        {
            SceneLoader.LoadScene(SceneType.HomeScreen);
        }

        public void Open(Dictionary<string, object> parameters)
        {
            bool ifWinner = (bool)parameters["GameResult"];

            TextType textType = ifWinner ? TextType.WinnerPlane :
    TextType.LooserPlane;

            string text = _gameDtaHolder.GetText(textType);

            _view.ShowText(text);

        }

        public void Dispose()
        {
            _view.onClose -= LoadMainScene;
            _view.onRetry -= ReloadScene;
        }
    }

    public class ProfileController : IController, IDisposable
    {
        private ProgressHandler _progress;

        private ProfileView _view;
```

```
    private IViewPool _windowPool;

    public ProfileController(ProgressHandler progress, IViewPool windowPool)
    {
        _windowPool = windowPool;
        _progress = progress;

        _view = (ProfileView)_windowPool.GetView(WindowType.ProfileWindow);

        _view.onClose += Close;
    }

    public void Open(Dictionary<string, object> parameters = null)
    {
        _view = (ProfileView)_windowPool.GetView(WindowType.ProfileWindow);

        _view.SetPlaneProgress(_progress.progress.plane.wins,
_progress.progress.plane.looses);

        _view.SetSpaceProgress(_progress.progress.space.wins,
_progress.progress.space.looses);
    }

    public void Close()
    {
        _windowPool.Push(WindowType.ProfileWindow);
    }

    public void Dispose()
    {
        _view.onClose -= Close;
    }
}

public class SpaceController: IController, IDisposable
{
    private SpaceView _view;

    private SpaceInfo _spaceInfo;

    private SpaceshipInfo _ship;

    private PlanetInfo _planet;

    private IViewPool _windowPool;

    private int _forceInfluenceDuration;

    private AsyncProcessor _asyncProcessor;

    private IThrownBodyPhysics _basePhysics;

    private IForceImpulsePhysics _forcePhysics;

    private ProgressHandler _progressHandler;

    private ISpaceDataHolder _spaceDataHolder;

    private IDistanceScale _scale;
```

```
    private float _distance;

    private float _time=0;

    public SpaceController(AsyncProcessor asyncProcessor, IViewPool windowPool,
ProgressHandler progressHandler, ISpaceDataHolder spaceDataHolder,
IForceImpulsePhysics forceImpulsePhysics, IThrownBodyPhysics thrownBodyPhysics,
IDistanceScale scale)
    {
        _windowPool = windowPool;
        _asyncProcessor = asyncProcessor;
        _progressHandler = progressHandler;
        _spaceDataHolder = spaceDataHolder;
        _scale = scale;

        _basePhysics = thrownBodyPhysics;
        _forcePhysics = forceImpulsePhysics;

        _view = (SpaceView)_windowPool.GetView(WindowType.SpaceMainWindow);

        _view.onFlightStart += ImitateFlight;

        _spaceInfo = _spaceDataHolder.GetSpaceInfo();

        RandomizeValues();

        CountDestination();
    }

    private void ImitateFlight(int angle, int force)
    {
        float speed = _forcePhysics.GetSpeedDiffByForce(force,
_forceInfluenceDuration, _ship.mass);
        _view.StartMovementAnimation();
        _asyncProcessor.StartCoroutine(FlyCoroutine(angle, speed));
    }

    private IEnumerator FlyCoroutine(int angle, float speed)
    {
        Vector3 indent;
        float distance = _basePhysics.GetFlightLength(speed, angle, _planet.g);
        do
        {
            _time += Time.deltaTime*10;
            indent = _basePhysics.GetLocationByTime(speed, angle, _planet.g,
_time);

            float rotation = _basePhysics.GetRotationByTime(speed, angle,
_planet.g, _time);

            _view.SetRocketPosition(indent*_scale.coeff, rotation);

            yield return null;

        } while (indent.x< distance);

        bool isWin = CheckDistanceIndent(distance);
        _progressHandler.AddSpacePoints(isWin ? 1 : -1);
        _view.StopMovementAnimation();

        OpenRetryWindow(isWin);
```

```
    }

    private void OpenRetryWindow(bool isWinner)
    {
        IController retryController =
_windowPool.GetController(WindowType.SpaceRetryWindow);

        Dictionary<string, object> parameters = new Dictionary<string,
object>();

        parameters.Add("GameResult", isWinner);

        retryController.Open(parameters);
    }

    private bool CheckDistanceIndent(float currentDistance)
    {
        return currentDistance >= _distance - _spaceInfo.touchdownErrorLimits.x
&&
            currentDistance <= _distance + _spaceInfo.touchdownErrorLimits.y;
    }

    private void RandomizeValues()
    {
        _ship =
_spaceDataHolder.GetRandomSpaceship(_progressHandler.progress.space.wins);
        _planet =
_spaceDataHolder.GetRandomPlanet(_progressHandler.progress.space.wins);
        _forceInfluenceDuration =
UnityEngine.Random.Range(_spaceInfo.minTimeForce, _spaceInfo.maxTimeForce);
        _view.Initialize(_spaceInfo, _ship, _planet, _forceInfluenceDuration);
    }

    private void CountDestination()
    {
        float force = UnityEngine.Random.Range(_spaceInfo.minForce,
_spaceInfo.maxForce + 1);
        float angle = UnityEngine.Random.Range(_spaceInfo.minAngle,
_spaceInfo.maxAngle + 1);

        float speed = _forcePhysics.GetSpeedDiffByForce(force,
_forceInfluenceDuration, _ship.mass);
         _distance = _basePhysics.GetFlightLength(speed, angle, _planet.g);

        _view.SetDistance(_distance, _scale.coeff, 1);
    }

    public void Open(Dictionary<string, object> parameters = null)
    {

    }

    public void Dispose()
    {
        _view.onFlightStart -= ImitateFlight;
    }
}

using System;
using System.Collections.Generic;
```

```
public class SpaceRetryWindowController : IController, IDisposable
{
    private RetryWindow _view;

    private IViewPool _windowPool;
    private IGameDataHolder _gameDataHolder;

    public SpaceRetryWindowController(IViewPool windowPool, IGameDataHolder
gameDataHolder)
    {
        _windowPool = windowPool;
        _gameDataHolder = gameDataHolder;

        _view = (RetryWindow) _windowPool.GetView(WindowType.SpaceRetryWindow);

        _view.onClose += LoadMainScene;
        _view.onRetry += ReloadScene;
    }

    private void ReloadScene()
    {
        _windowPool.Push(WindowType.SpaceRetryWindow);
        SceneLoader.LoadScene(SceneType.RocketGame);
    }

    private void LoadMainScene()
    {
        SceneLoader.LoadScene(SceneType.HomeScreen);
    }

    public void Open(Dictionary<string, object> parameters)
    {
        bool ifWinner = (bool)parameters["GameResult"];

        TextType textType = ifWinner ? TextType.WinnerSpace :
TextType.LooserSpace;

        string text= _gameDataHolder.GetText(textType);

        _view.ShowText(text);

    }

    public void Dispose()
    {
        _view.onClose -= LoadMainScene;
        _view.onRetry -= ReloadScene;
    }
}

public class LocalDataController:ILocalDataController
{
    public string Serialize<T>(T jsonObject)
    {
        string json = JsonConvert.SerializeObject(jsonObject);
        return json;
    }

    public T Deserialize<T>(string data)
    {
        T getObject = JsonConvert.DeserializeObject<T>(data);
```

```
        return getObject;
    }

    public string LoadData(string nameFile, bool useEncryption = false)
    {
        string finalPath = Path.Combine(Application.persistentDataPath,
nameFile);

        string result = string.Empty;

        if (File.Exists(finalPath))
        {
            result = File.ReadAllText(finalPath);
        }

        return result;
    }

    public string LoadDataPath(string path, bool useEncryption = false)
    {
        string finalPath = path;

        string result = string.Empty;

        if (File.Exists(finalPath))
        {
            result = File.ReadAllText(finalPath);
        }

        return result;
    }

    public string LoadDataFromAssets(string nameFile, bool useEncryption =
false)
    {
        string path = Path.Combine(Application.dataPath, "Data");

        string finalPath = Path.Combine(path, nameFile);

        string result = string.Empty;

        if (File.Exists(finalPath))
        {
            result = File.ReadAllText(finalPath);
        }

        return result;
    }

    public void SaveData(string namefile, string data)
    {
        string path = Path.Combine(Application.persistentDataPath);

        string finalPath = Path.Combine(path, namefile);

        if (!Directory.Exists(path))
        {
            Directory.CreateDirectory(path);
        }

        using (FileStream fs = File.Create(finalPath))
```

```
        {
            AddText(fs, data);
        }
    }

    public void SaveData<T>(T jsonObject, string nameFile)
    {
        string path = Path.Combine(Application.persistentDataPath);

        string finalPath = Path.Combine(path, nameFile);

        if (!Directory.Exists(path))
        {
            Directory.CreateDirectory(path);
        }

        string data = Serialize(jsonObject);
        using (FileStream fs = File.Create(finalPath))
        {
            AddText(fs, data);
        }
    }

    private void AddText(FileStream fs, string value)
    {
        byte[] info = new UTF8Encoding(true).GetBytes(value);

        fs.Write(info, 0, info.Length);
    }

}

public class GameDataHolder:IGameDataHolder
{
    private GameTexts _texts;

    public GameDataHolder()
    {
        LoadAssets();
    }

    public string GetText(TextType type)
    {
        return _texts.gameTexts.FirstOrDefault(t=>t.type==type).phrase;
    }


    private void LoadAssets()
    {
        _texts = (GameTexts)Resources.Load("GameTexts", typeof(GameTexts));
    }

}

public class PlaneDataHolder:IPlaneDataHolder
{
    private List<PlaneMaterial> _materials = new List<PlaneMaterial>();

    private PlaneInfo _planeInfo;

    public PlaneDataHolder()
```

```
    {
        LoadAssets();
    }

    public PlaneMaterial GetRandomMaterial(int wins)
    {
        return _materials.Where(s => s.wins <= wins).ToList().GetRandom();
    }

    public PlaneMaterial GetMaterial(string name)
    {
        return _materials.FirstOrDefault(s => s.name == name);
    }

    public List<PlaneMaterial> GetMaterials()
    {
        return _materials.OrderBy(s=>s.wins).ToList();
    }

    public PlaneInfo GetPlaneInfo()
    {
        return _planeInfo;
    }

    private void LoadAssets()
    {
        foreach (PlaneMaterial conf in Resources.LoadAll("Materials",
typeof(PlaneMaterial)))
        {
            _materials.Add(conf);
        }

        _planeInfo = (PlaneInfo)Resources.Load("PlaneInfo", typeof(PlaneInfo));
    }
}

public class PrefabsHolder:IPrefabsHolder
{
    private PrefabsInfo _prefabs;

    public PrefabsHolder()
    {
        LoadAssets();
    }

    public T GetPrefab<T>() where T:BasePrefab
    {
        var items = _prefabs.prefabs.OfType<T>().ToList();
        var item = (T)items[0];
         return item;
    }


    public void LoadAssets()
    {
        _prefabs = (PrefabsInfo)Resources.Load("Prefabs", typeof(PrefabsInfo));
    }
}

public class ScalesHolder:IScalesHolder
{
```

```
    private ScaleInfo _scale;

     public ScalesHolder()
    {
        LoadAssets();
    }


    public PlaneScale GetPlaneScale()
    {
        return _scale.planeScale;
    }

    public RocketScale GetRocketScale()
    {
        return _scale.rocketScale;
    }

    private void LoadAssets()
    {
        ScalesInfo scales = (ScalesInfo)Resources.Load("ScalesInfo",
typeof(ScalesInfo));
        float ratio = CountRatio();

        _scale = scales.scales.FirstOrDefault(s => s.sizeLimits.x < ratio &&
s.sizeLimits.y > ratio);
    }

    private float CountRatio()
    {
        float height = Camera.main.orthographicSize * 2.0f;
        float width = height * Screen.width / Screen.height;

        return height / width;
    }
}

public class SpaceDataHolder:ISpaceDataHolder
{
    private List<PlanetInfo> _planets=new List<PlanetInfo>();

    private List<SpaceshipInfo> _spaceShips = new List<SpaceshipInfo>();

    private SpaceInfo _spaceInfo;

    public SpaceDataHolder()
    {
        LoadAssets();
    }

    public PlanetInfo GetRandomPlanet(int wins)
    {
        return _planets.Where(s => s.wins <= wins).ToList().GetRandom();
    }

    public PlanetInfo GetPlanet(string name)
    {
        return _planets.FirstOrDefault(p => p.name == name);
    }

    public List<PlanetInfo> GetPlanets()
```

```
    {
        return _planets.OrderBy(p => p.wins).ToList();
    }

    public SpaceshipInfo GetRandomSpaceship(int wins)
    {
        return _spaceShips.Where(s=>s.wins<=wins).ToList().GetRandom();
    }

    public SpaceshipInfo GetSpaceship(string name)
    {
        return _spaceShips.FirstOrDefault(s => s.name == name);
    }

    public float GetMinSpaceshipMass()
    {
        return _spaceShips.Select(s=>s.mass).Min();
    }

    public List<SpaceshipInfo> GetSpaceships()
    {
        return _spaceShips.OrderBy(s => s.wins).ToList();
    }

    public SpaceInfo GetSpaceInfo()
    {
        return _spaceInfo;
    }

    private void LoadAssets()
    {
        foreach (PlanetInfo conf in Resources.LoadAll("Planets",
typeof(PlanetInfo)))
        {
            _planets.Add(conf);
        }

        foreach (SpaceshipInfo conf in Resources.LoadAll("Spaceships",
typeof(SpaceshipInfo)))
        {
            _spaceShips.Add(conf);
        }

        _spaceInfo = (SpaceInfo) Resources.Load("SpaceInfo", typeof(SpaceInfo));
    }

    public float GetMinG()
    {
        return _planets.Select(p => p.g).Min();
    }
}

public static class Extentions
{
    public static T GetRandom<T>(this List<T> elements)
    {
        return elements[elements.Count > 1 ? UnityEngine.Random.Range(0,
elements.Count) : 0];
    }

}
```

```
public class GameInstaller : MonoInstaller
{
    [SerializeField]
    private AsyncProcessor _asyncProcessor;

    public override void InstallBindings()
    {
        Debug.Log("GameInstaller");
        Container.BindInstance(_asyncProcessor);


Container.Bind(typeof(ProgressHandler)).To<ProgressHandler>().AsSingle();

        Container.Bind<IPrefabsHolder>().To<PrefabsHolder>().AsSingle();


Container.Bind<ILocalDataController>().To<LocalDataController>().AsSingle();

        Container.Bind<IGameDataHolder>().To<GameDataHolder>().AsSingle();
        Container.Bind<IScalesHolder>().To<ScalesHolder>().AsSingle();

        Container.Bind<IPlaneDataHolder>().To<PlaneDataHolder>().AsSingle();
        Container.Bind<ISpaceDataHolder>().To<SpaceDataHolder>().AsSingle();
    }

}

public class HomeInstaller : MonoInstaller<HomeInstaller>, IControllerFactory
{
    [SerializeField]
    private List<Window> _windows;

    public override void InstallBindings()
    {
        Container.BindInstance(_windows);

Container.Bind<IViewPool>().To<ViewPool>().AsSingle().WithArguments(_windows);

        BindControllers();


Container.Bind(typeof(IControllerFactory)).FromInstance(this).AsSingle();

        Container.Bind<HomeLoader>().AsSingle().NonLazy();

    }

    public  IController CreateController(WindowType type)
    {
        switch (type)
        {
            case WindowType.HomeWindow:
                return Container.Resolve<HomeController>();

            case WindowType.ProfileWindow:
                return Container.Resolve<ProfileController>();

            case WindowType.RocketsWindow:
                return Container.Resolve<AvailableRocketsController>();
```

```
            case WindowType.PlanetsWindow:
                return Container.Resolve<AvailablePlanetsController>();

            case WindowType.MaterialsWindow:
                return Container.Resolve<AvailablePlanesController>();

            default:
                return null;
        }
    }

    public void BindControllers()
    {
        Container.Bind(typeof(HomeController)).To<HomeController>().AsSingle();


Container.Bind(typeof(ProfileController)).To<ProfileController>().AsSingle();


Container.Bind(typeof(AvailableRocketsController)).To<AvailableRocketsController
> ().AsSingle();


Container.Bind(typeof(AvailablePlanetsController)).To<AvailablePlanetsController
>().AsSingle();


Container.Bind(typeof(AvailablePlanesController)).To<AvailablePlanesController>(
).AsSingle();

    }

}


public class PlaneInstaller : MonoInstaller, IControllerFactory
{
    [SerializeField]
    private List<Window> _windows;

    public override void InstallBindings()
    {
        Container.BindInstance(_windows);

Container.Bind<IViewPool>().To<ViewPool>().AsSingle().WithArguments(_windows);

Container.Bind<IInclinedPlanePhysics>().To<InclinedPlanePhysics>().AsSingle();

        BindControllers();

Container.Bind(typeof(IControllerFactory)).FromInstance(this).AsSingle();

        Container.Bind<PlaneLoader>().AsSingle().NonLazy();
    }

    public void BindControllers()
    {

Container.Bind(typeof(PlaneRetryWindowController)).To<PlaneRetryWindowController
>().AsSingle();
```

```
Container.Bind(typeof(PlaneController)).To<PlaneController>().AsSingle();
    }

    public IController CreateController(WindowType type)
    {
        switch (type)
        {
           case WindowType.PlaneRetryWindow:
                return Container.Resolve<PlaneRetryWindowController>();

           case WindowType.PlaneMainWindow:
                return Container.Resolve<PlaneController>();

           default:
                return null;
        }
    }

}

public class SpaceInstaller : MonoInstaller, IControllerFactory
{
    [SerializeField]
    private List<Window> _windows;

    public override void InstallBindings()
    {
        Container.BindInstance(_windows);

Container.Bind<IViewPool>().To<ViewPool>().AsSingle().WithArguments(_windows);
        Container.Bind<IThrownBodyPhysics>().To<ThrownBodyPhysics>().AsSingle();

Container.Bind<IForceImpulsePhysics>().To<ForceImpulsePhysics>().AsSingle();

Container.Bind<IDistanceScale>().To<DistanceScale>().AsSingle().NonLazy();

        BindControllers();

Container.Bind(typeof(IControllerFactory)).FromInstance(this).AsSingle();

        Container.Bind<SpaceLoader>().AsSingle().NonLazy();
    }

    public void BindControllers()
    {

Container.Bind(typeof(SpaceRetryWindowController)).To<SpaceRetryWindowController
>().AsSingle();


Container.Bind(typeof(SpaceController)).To<SpaceController>().AsSingle();
    }

    public IController CreateController(WindowType type)
    {
        switch (type)
        {
            case WindowType.SpaceRetryWindow:
                return Container.Resolve<SpaceRetryWindowController>();
```

```
            case WindowType.SpaceMainWindow:
                return Container.Resolve<SpaceController>();

            default:
                return null;
        }
    }

}

public class ForceImpulsePhysics:IForceImpulsePhysics
{
    public float GetSpeedDiffByForce(float force, float duration, float mass)
    {
        return (force * duration) / mass;
    }
}

public class InclinedPlanePhysics: IInclinedPlanePhysics
{
   public float GetDistance(float a, float t)
    {
        return a * t * t * 0.5f;
    }

    public float GetTime(float s, float a)
    {
        return (float)Math.Sqrt((2 * s) / a);
    }

    public float GetAcceleration(float g, float angle, float mu, float m)
    {
        float forces = g * (float)(Math.Sin(angle * Mathf.Deg2Rad) + mu *
Math.Cos(angle * Mathf.Deg2Rad));
        return forces;
    }

    public float GetSpeed(float a, float t, float v0)
    {
        return v0 + a * t;
    }
}

public class ThrownBodyPhysics:IThrownBodyPhysics
{
    public Vector3 GetLocationByTime(float speed, float angle, float g, float
time)
    {
        float x = speed * (float)Math.Cos(angle * Mathf.Deg2Rad) * time;
        float y = (speed * (float)Math.Sin(angle * Mathf.Deg2Rad) * time - g *
time * time * 0.5f);

        return new Vector3(x, y);
    }

    public float GetRotationByTime(float speed, float angle, float g, float
time)
    {
        var vx = speed * (float)Math.Cos(angle * Mathf.Deg2Rad);
        var vy = speed * (float)Math.Sin(angle * Mathf.Deg2Rad) - g * time;
```

```
            var tg = vy / vx;
            var rotation = (float)Math.Atan(tg) * Mathf.Rad2Deg;

            return rotation;
        }

        public float GetFlightLength(float speed, float angle, float g)
        {
            return (speed * speed * (float)Math.Sin(2 * angle * Mathf.Deg2Rad)) / g;
        }
    }

    public class Progress
    {
        [JsonProperty("Space")]
        public Points space;

        [JsonProperty("Plane")]
        public Points plane;
    }

    public class Points
    {
        [JsonProperty("wins")]
        public int wins;

        [JsonProperty("looses")]
        public int looses;

        [JsonConstructor]
        public Points(int wins, int looses)
        {
            this.wins = wins;
            this.looses = looses;
        }
    }

    using System;
    public class ProgressHandler
    {
        private Progress _progress;

        private ILocalDataController _localDataController;

        public Progress progress => _progress;

        public ProgressHandler(ILocalDataController localDataController)
        {
            _localDataController = localDataController;

            Initialize();
        }

        private void Initialize()
        {
            string dataProgress = _localDataController.LoadData("Progress");

            _progress = string.IsNullOrEmpty(dataProgress) ? FillBaseProgress() :
    _localDataController.Deserialize<Progress>(dataProgress);
        }
```

```
    private Progress FillBaseProgress()
    {
        return new Progress()
        {
            space = new Points(0,0),
            plane =new Points(0,0)
        };
    }

    private void Save()
    {
        _localDataController.SaveData(_progress, "Progress");
    }

    public void AddSpacePoints(int points)
    {
        if(points>0)
        {
            _progress.space.wins += points;
        }

        if (points < 0)
        {
            _progress.space.looses -= points;
        }

        Save();
    }

    public void AddPlanePoints(int points)
    {
        if (points > 0)
        {
            _progress.plane.wins += points;
        }

        if (points < 0)
        {
            _progress.plane.looses -= points;
        }

        Save();
    }

}

public class DistanceScale:IDistanceScale
{
    private float _coeff;

    public float coeff => _coeff;

    public DistanceScale(IScalesHolder scalesHolder, ISpaceDataHolder
spaceDataHolder, IForceImpulsePhysics forceImpulsePhysics, IThrownBodyPhysics
thrownBodyPhysics)
    {
        float scaleFactor = scalesHolder.GetRocketScale().widthScale;

        var spaceInfo = spaceDataHolder.GetSpaceInfo();
```

```
        float speed =
forceImpulsePhysics.GetSpeedDiffByForce(spaceInfo.maxForce,
spaceInfo.maxTimeForce, spaceDataHolder.GetMinSpaceshipMass());

        float maxDistance = thrownBodyPhysics.GetFlightLength(speed, 45,
spaceDataHolder.GetMinG());

        _coeff = scaleFactor / maxDistance;

    }
}

public class ViewPool:IViewPool
{
    private List<Window> _windowPrefabs;

    private Dictionary<WindowType, WindowView> _viewPool= new
Dictionary<WindowType, WindowView>();

    private IControllerFactory _controllerFactory;

    public ViewPool(IControllerFactory controllerFactory, List<Window>
windowPrefabs)
    {
        _windowPrefabs = windowPrefabs;
       _controllerFactory = controllerFactory;
    }

    public IController GetController(WindowType type)
    {
        return _controllerFactory.CreateController(type);
    }

    public WindowView GetView(WindowType type)
    {
        if (!_viewPool.ContainsKey(type))
        {
            var prefab = _windowPrefabs.FirstOrDefault(w => w.type ==
type).prefab;
            _viewPool.Add(type, GameObject.Instantiate(prefab));
            _viewPool[type].gameObject.SetActive(false);
        }
        _viewPool[type].gameObject.SetActive(true);
        return _viewPool[type];
    }

    public void Push(WindowType type)
    {
        _viewPool[type].gameObject.SetActive(false);
    }
}

[Serializable]
public class Window
{
    [SerializeField]
    public WindowType _type;

    [SerializeField]
    public WindowView _prefab;
```

```
    public WindowType type { get { return _type; } }

    public WindowView prefab { get { return _prefab; } }
}

public enum WindowType
{
    SpaceRetryWindow=0,
    PlaneRetryWindow=1,
    ProfileWindow=2,
    RocketsWindow=3,
    MaterialsWindow=4,
    SpaceMainWindow=5,
    PlaneMainWindow=6,
    HomeWindow=7,
    PlanetsWindow=8
}

public class AsyncProcessor : MonoBehaviour
{

}

public static class SceneLoader
{
    public static void LoadScene(SceneType name)
    {
        SceneManager.LoadScene(name.ToString());
    }
}

public enum SceneType
{
    RocketGame,
    PlaneGame,
    HomeScreen
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

Програма та методика тестування

КПІ.ІП-6115.045490.05.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Л.С. Кухарець

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний тренажер для вивчення взаємодії фізичних сил, що представляє із себе гру на смартфон або планшет з системою Android, створений за допомогою мови програмування C# та ігрового рушія Unity. з використанням ігрового рушія Unity.

2 МЕТА ТЕСТУВАННЯ

Під час тестування мають бути протестовані функціональні та інтерфейсні особливості ПЗ.

UI компоненти, що мають бути протестовані, включають:

– коректну реакцію клавіш на натискання;

– виведення потрібних екранів та вікон;

– коректне відображення даних у текстових полях;

– правильну реакція слайдерів на дії користувача.

Функціональні компоненти, що мають бути протестовані, включають:

– вхід в гру;

– вихід з гри;

– перехід на сцени ігор;

– збереження прогресу;

– відкривання нових параметрів ігор зі збільшенням виграшів.

Компоненти, що мають бути протестовані у грі про рух тіла, кинутого під кутом до горизонту:

– перезавантаження рівня;

– вихід в меню;

– програш чи перемога;

– можливість вибору кута та сили, прикладеної до об'єкта;

– рандомне задання параметрів задачі;

– слідування об’єктом траєкторії польоту.

Компоненти, що мають бути протестовані у грі про рух тілапохилою площиною:

– перезавантаження рівня;

– вихід в меню;

– програш чи перемога;

- можливість вибору кута площини та маси об'єкта;
- рандомне задання параметрів задачі;
- слідування зміни швидкості об’єкта фізичним формулам.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування для проекту складатиметься з наступних методів:

а) лог-тестування;

б) інтеграційне тестування;

в) системне тестування:

1) функціональне тестування;

2) UI (User Interface) тестування.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

а) Лог-тестування.

Буде виконане за допомогою виведення повідомлень про помилку та обраних проміжних значень у лог-консоль редактору Unity.

б) Інтеграційне тестування.

Буде виконане шляхом збірки та тестування кожної з міні-ігор окремо, а також збірки та тестування усього програмного забезпечення за умови, якщо ігри функціонують коректно.

в) Системне тестування.

1) Функціональне тестування.

Буде виконано мануально. Окрім перевірки функціональної відповідності програми вимогам, тестувальник має перевірити коректність розв’язання програмою задач шляхом паралельного розв’язання задач з такими ж параметрами за допомогою фізичного калькулятора.

2) UI тестування.

Виконуватиметься паралельно із попереднім етапом. Має перевірити коректну роботу кнопок, слайдерів, скрол-барів тощо.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

____Олександр ПАВЛОВ

“___” _____ 2020 р.

МОБІЛЬНИЙ ТРЕНАЖЕР ДЛЯ ВИВЧЕННЯ ВЗАЄМОДІЇ ФІЗИЧНИХ СИЛ З ВИКОРИСТАННЯМ ІГРОВОГО РУШІЯ UNITY

Керівництво користувача

КПІ.ІП-6115.045490.06.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Л.С. Кухарець

Київ – 2020 року

В даному документі представлене керівництво користувача для роботи з програмним застосунком даного дипломного проекту.

Дії користувача у головному меню.

а) Зайшовши в гру, користувач одразу потрапляє в головне меню.

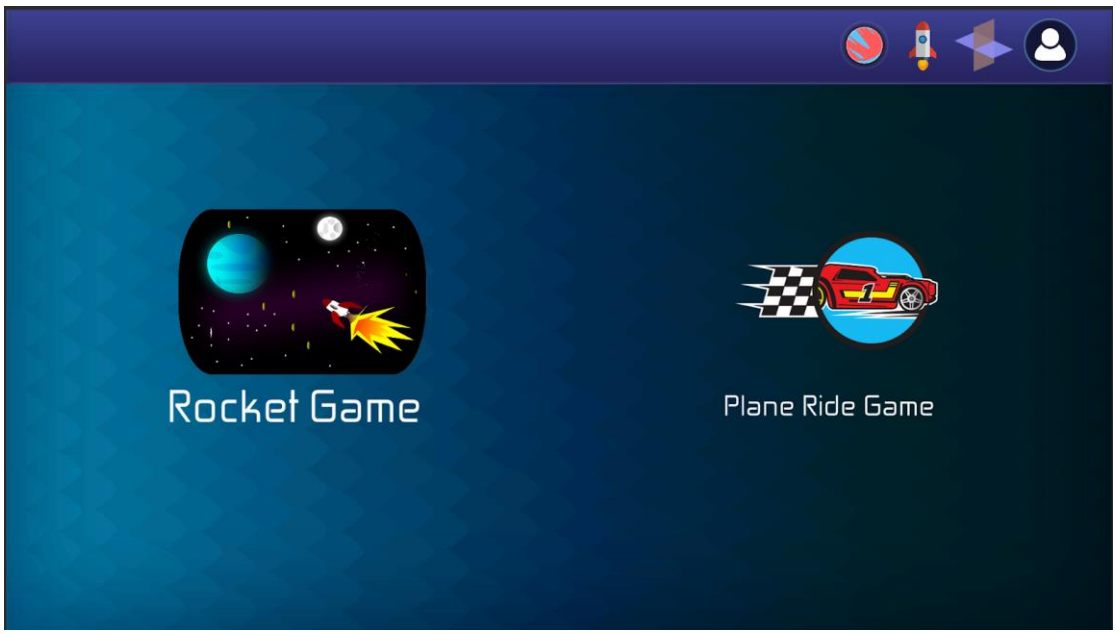


Рисунок 1 - Головне меню

б) Аби переглянути доступні планети, необхідно натиснути на графічне зображення планети на верхній панелі екрану. Представлені об’єкти можна скролити горизонтально. Для повернення в головне меню потрібно натиснути кнопку «Назад».

Рисунок 2 - Доступні планети

в) Аби переглянути доступні ракети, необхідно натиснути на графічне зображення ракети на верхній панелі екрану.

Рисунок 3 - Доступні ракети

в) Аби переглянути доступні матеріали площини, необхідно натиснути на графічне зображення площин на верхній панелі екрану.

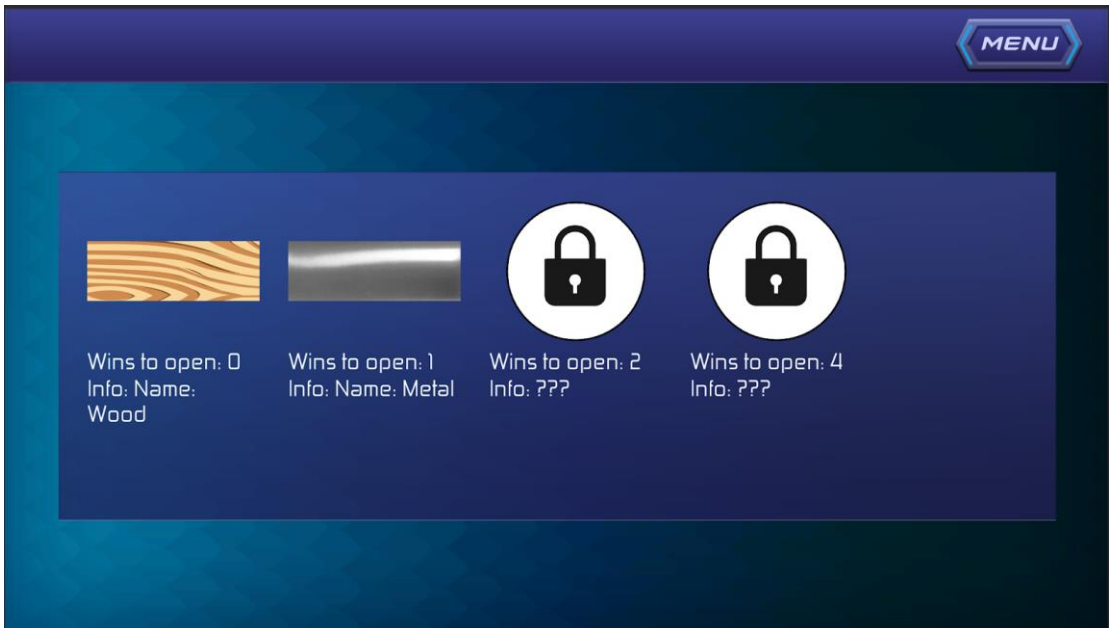


Рисунок. 4 - Доступні матеріали площини

г) Аби переглянути інформацію про програші та виграші, необхідно натиснути на графічне зображення іконки профілю на верхній панелі екрану.

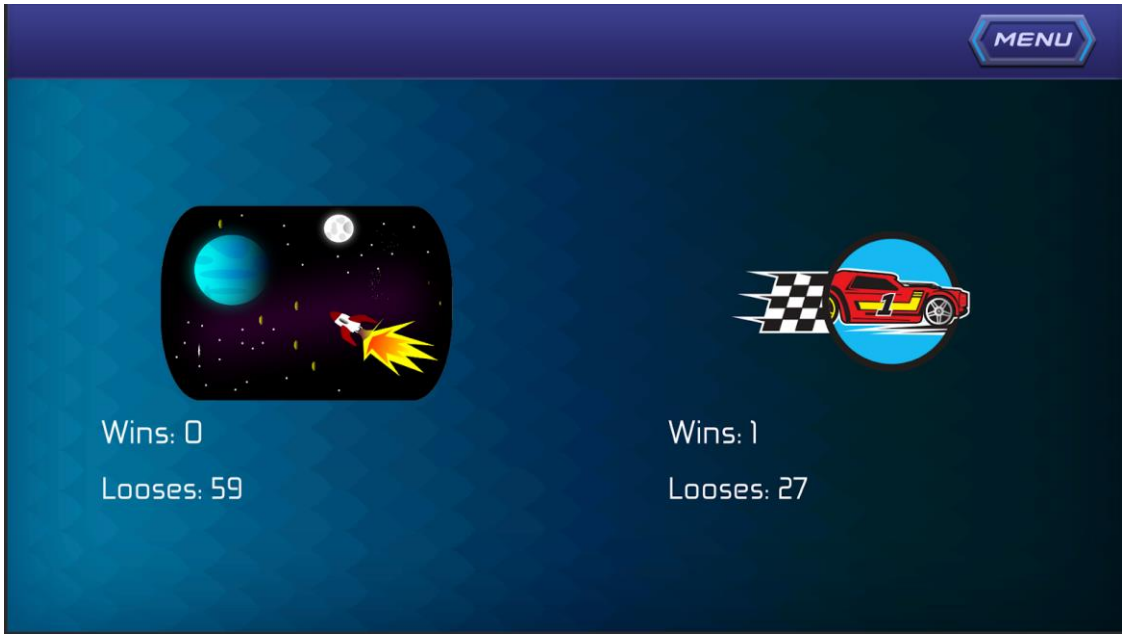


Рисунок 5 - Екран профілю

д) Для переходу на сцену міні гри, користувачу необхідно натиснути на іконку з зображенням космосу або автомобіля на екрані головного меню.

Дії користувача у грі про рух тіла, кинутого під кутом до горизонту.

а) При виборі гри про рух тіла, кинутого під кутом до горизонту завантажується сцена з космічною грою. Верхній скролер відповідає за налаштування сили, що буде прикладена до тіла. Нижній скролер регулює кут, під яким тіло полетить. У текстових панелях зазначно тривалість прикладання сили, масу об’єкту, назву планети та коефіцієнт прискорення вільного падіння.

Рисунок 6 - Гра про рух тіла, кинутого під кутом до горизонту

б) Якщо натиснути на кнопку польоту, ракета почне рух.

Рисунок 7 - Анімація руху ракети

в) Після приземлення ракети з’явиться вікно з інформацією про виграш чи програш. Натиснувши на праву клавішу, можна повернутися у головне меню, на ліву – зіграти ще раз.

Рисунок 8 - Вікно програшу чи виграшу

Дії користувача у грі про рух тіла похилою площиною.

а) При виборі гри про рух тілапохилою площиною завантажується сцена з грою про рух дерев'яного іграшкового автомобіля похилою поверхнею. Верхній скролер відповідає за налаштування маси автомобіля. Нижній скролер регулює кут, під яким нахилена площина. У текстових панелях зазначено тривалість матеріал площини, її довжину, коефіцієнт тертя між об'єктами, бажану кінцеву та поточну швидкості.

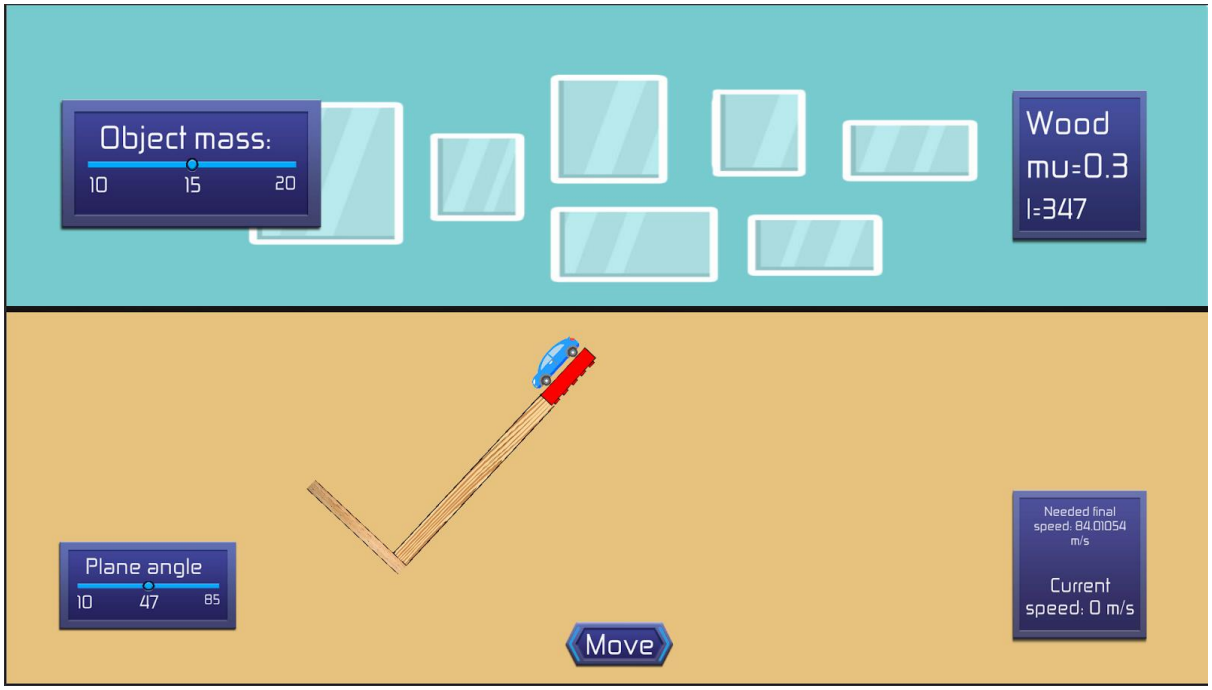


Рисунок 9 - Гра про рух тіла похилою площиною

б) Якщо натиснути на кнопку руху, автомобіль покотиться вниз.

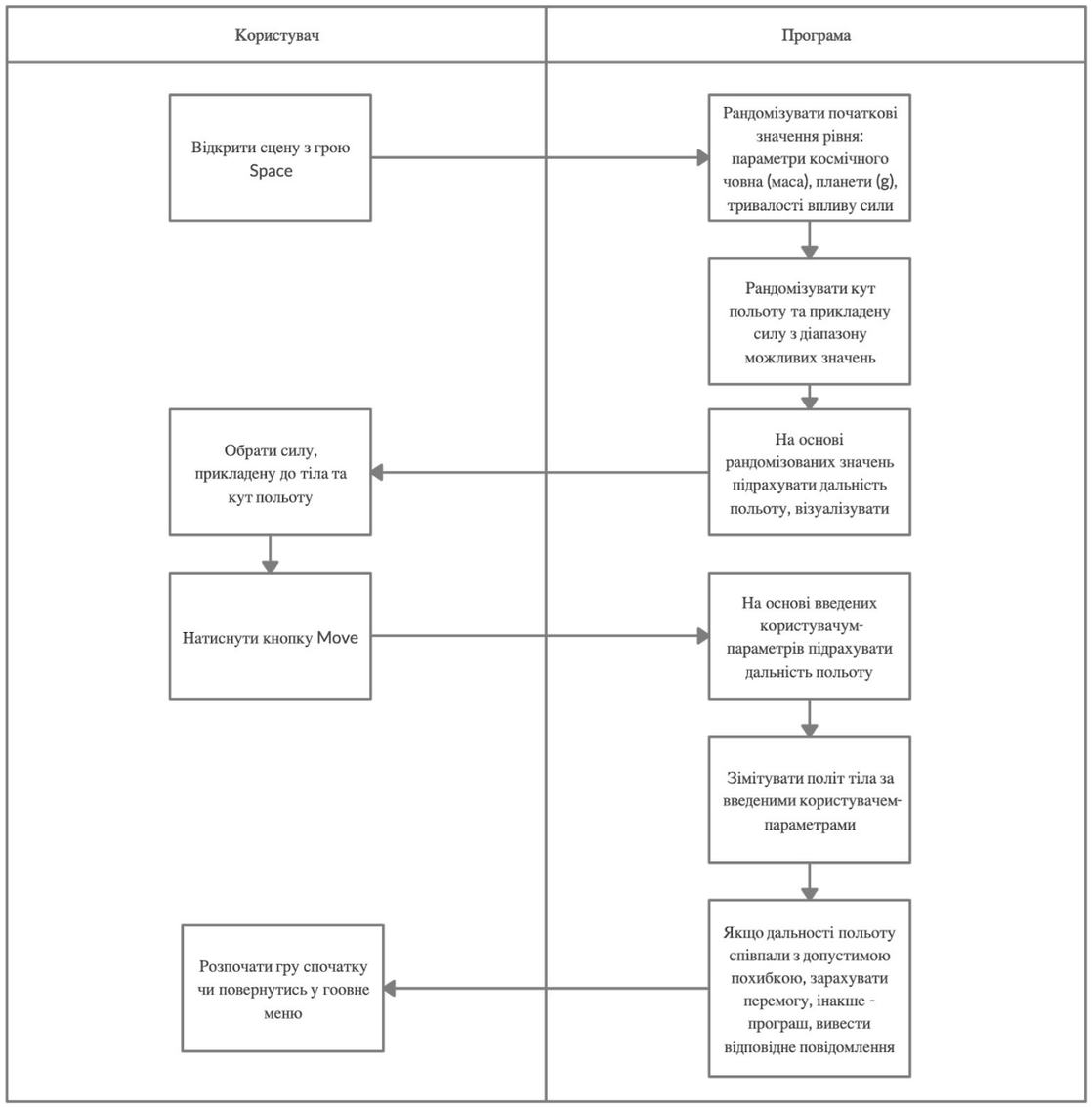
Рисунок 10 - Анімація руху тіла похилою площиною

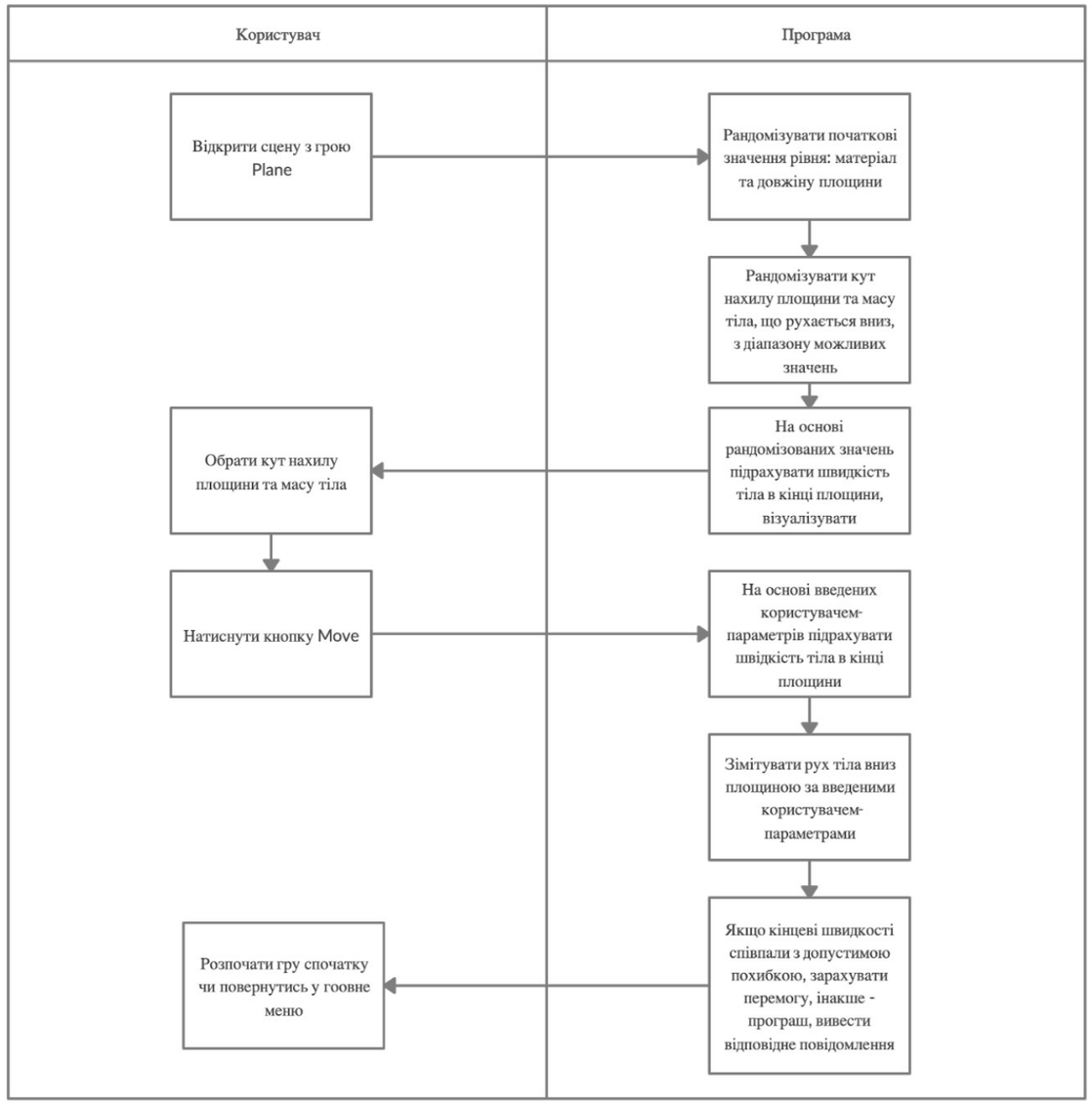
в) Після досягнення автомобілем фінальної точки з’явиться вікно з інформацією про виграш чи програш. Натиснувши на праву клавішу, можна повернутися у головне меню, на ліву – зіграти ще раз.

КПІ.ІП-6115.045490.07.99.СС

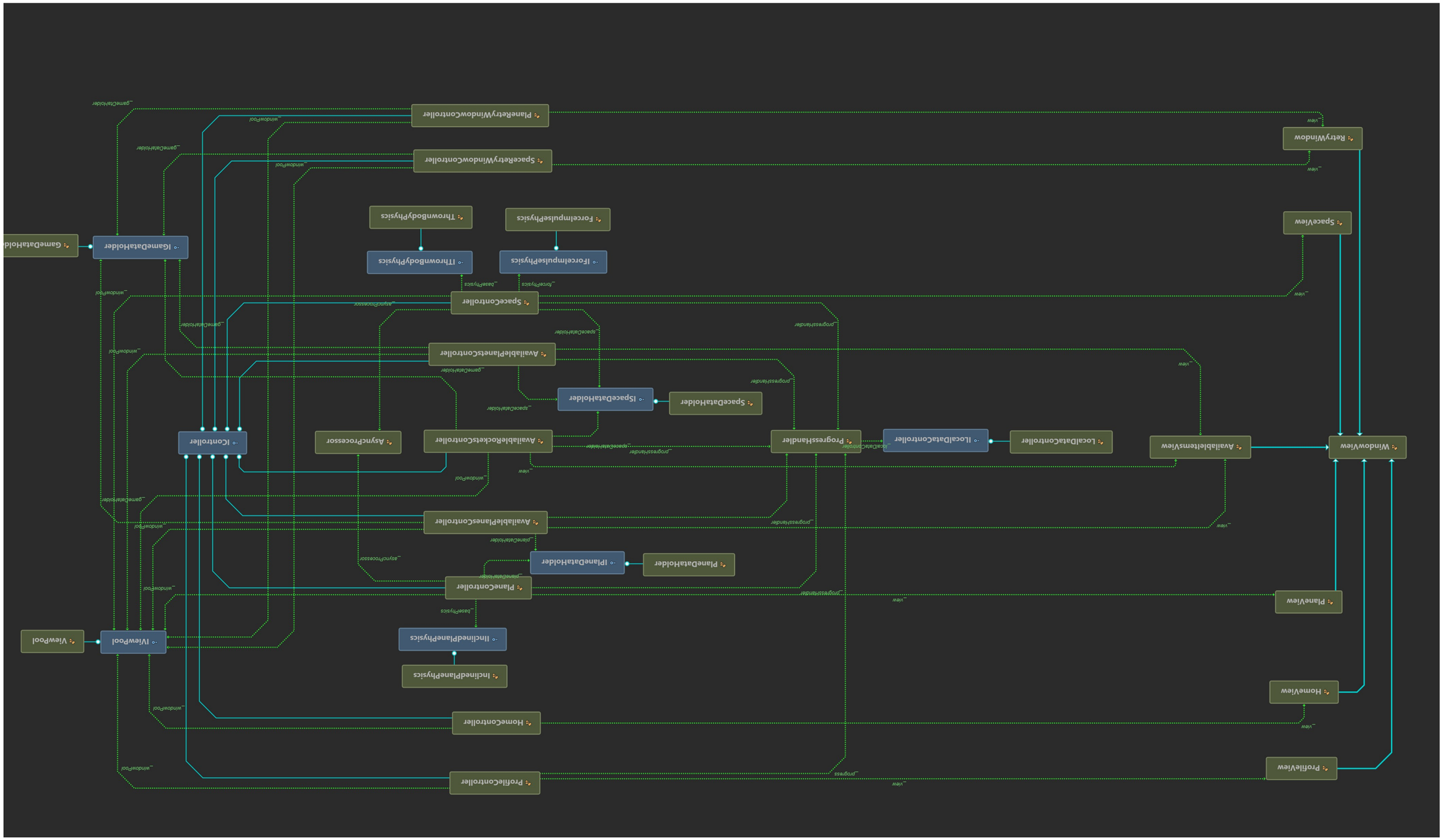
Інв. № підп.	Підп. дата	Інв. № взам. підп.	Інв. № дубл.	Підп. дата

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-6115.045490.07.99.СС		
Розроб.		Кухарець Л.С.			Схема структурна варіантів використання	Лит.	Арк.	Аркушів
Перев.		Ліщук К.І.						1
Т. Кон.						Аркуш		Аркушів
Н. Кон.		Ліщук К.І.			Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61		
Затв.		Ліщук К.І.						





					КПІ.ІП-6115.045490.07.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера / Маса / Масштаб
Розробив		Кухарець Л.С.				
Перевірив		Ліщук К.І.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Мобільний тренажер для вивчення взаємодії фізичних сил з використанням ігрового рушія Unity	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61
Затвердив		Ліщук К.І.				



Зм.	Арк.	№ документа	Підпис	Дата							
Розробив		Кухарець Л.С.									
Перевірив		Плаух К.І.									
Т. контр.											
Н. контр.		Плаух К.І.									
Затвердив		Плаух К.І.									

КПІ.ІП-6115.045490.07.99.СС

Схема структура класів програмного забезпечення

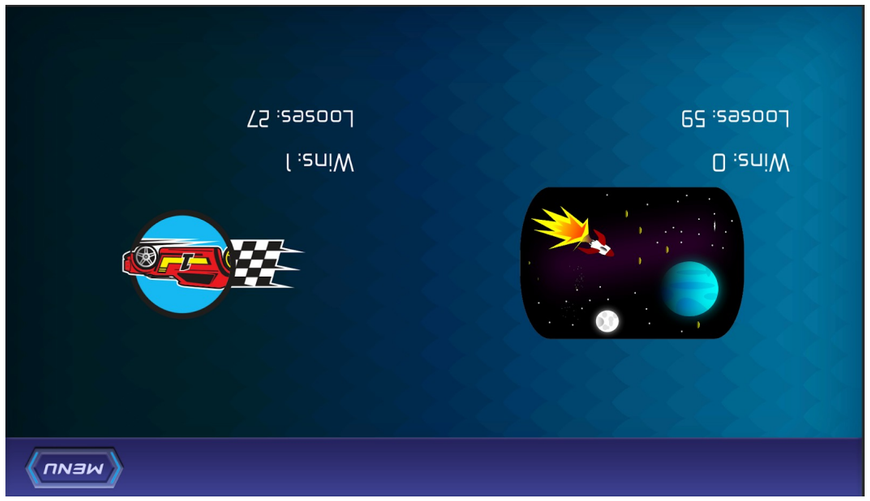
Літера	Маса	Масштаб

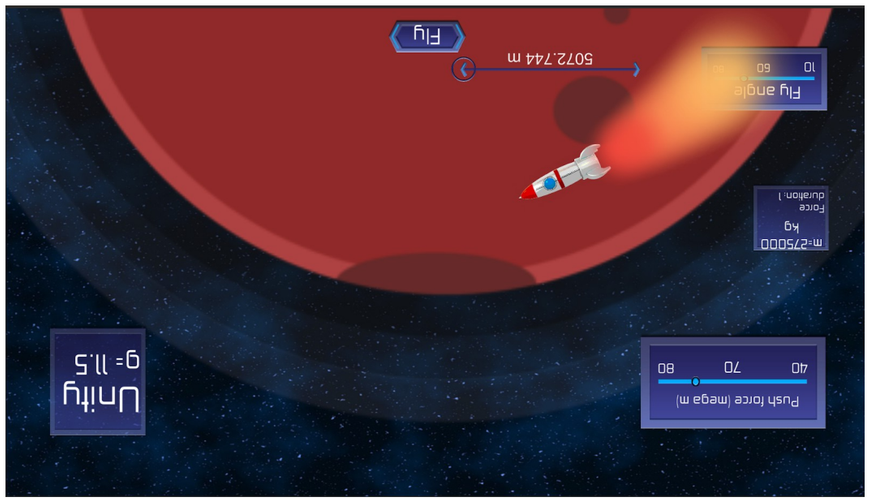
Аркуш	Аркушів

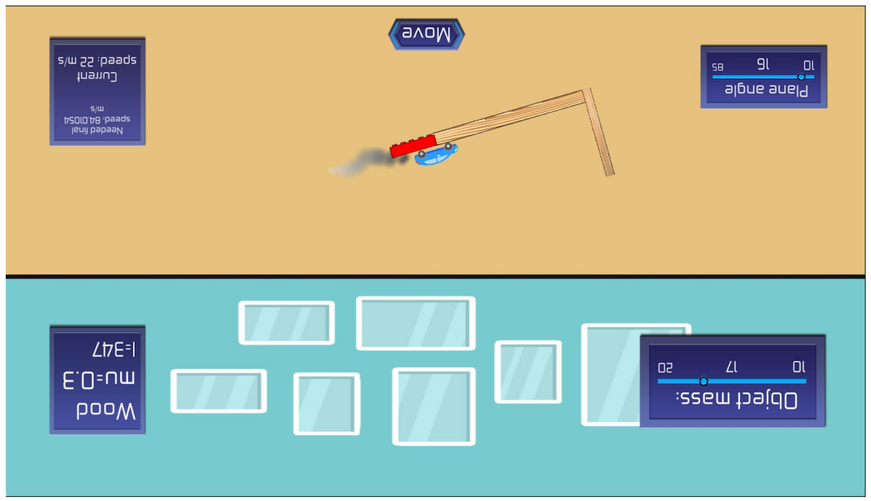
Мобільний менеджер для вдячення вдячності фізичних сил з використанням ігрового рушія Unity

КПІ ім.Ігоря Сікорського
Кафедра АСОІУ
гр. ІП-61

КПІ.ІП-6115.045490.07.99.СС







Зм.	Арк.	№ докумнта	Підпис	Дата	*КПІ.ІП-6115.045490.07.99.КЕ*					
Розробив		*Кухарець Л.С.*								
Перевірив		*Плиук К.І.*			*Кретсленна вказівку екранних форм*			Літера	Маса	Масштаб
Т. кон.										
								Аркуш	Аркушів	
Н. кон.		*Плиук К.І.*			*Мобільний менеджер для вцечення взаємодії фізичних сил з використанням ігрового рушія Unity*			КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61		
Затвердив		*Плиук К.І.*								